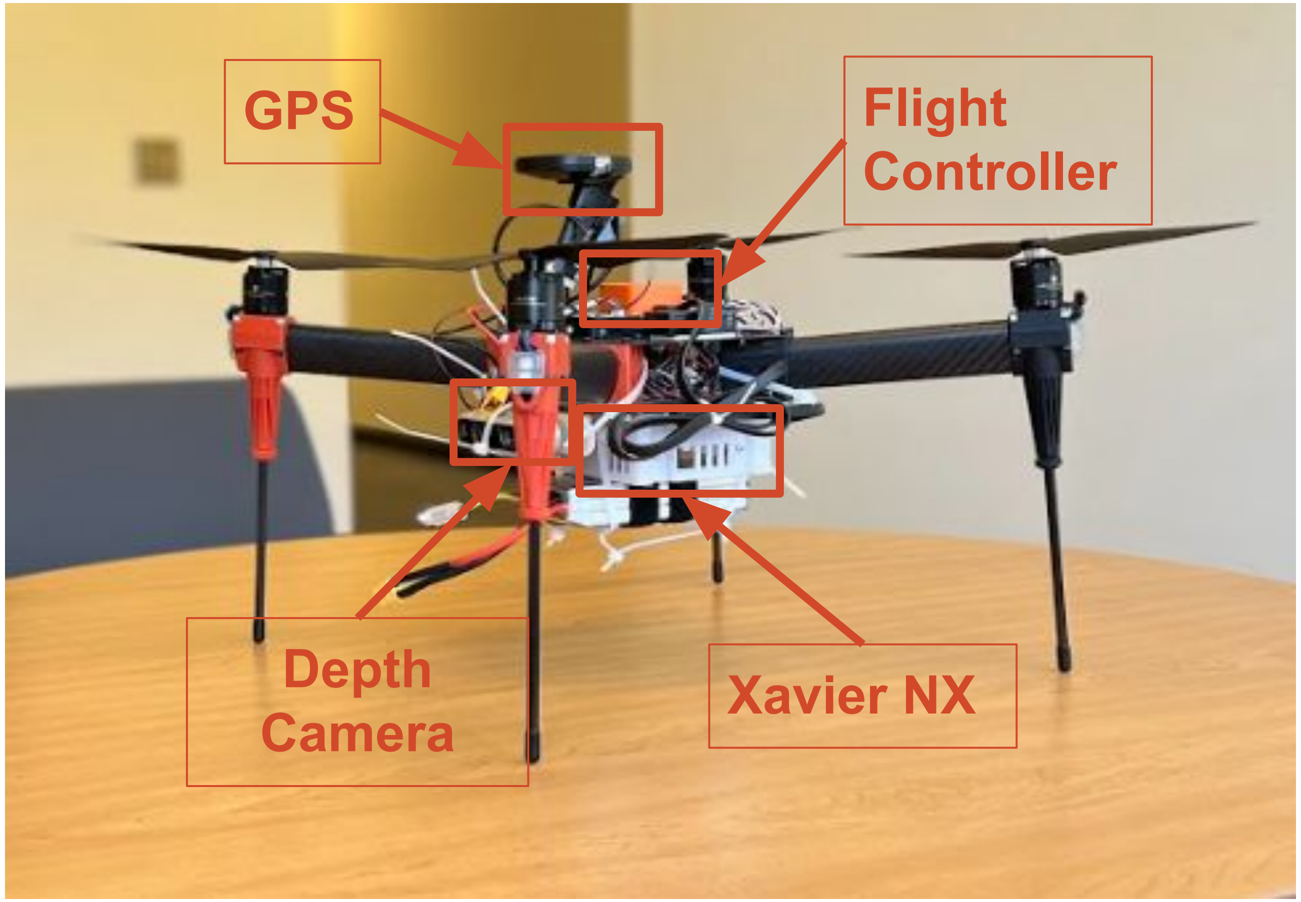
Context-aware Multi-Model Object Detection for Diversely Heterogeneous Compute Systems

Justin Davis
Mehmet E. Belviranli

Introduction

System Overview

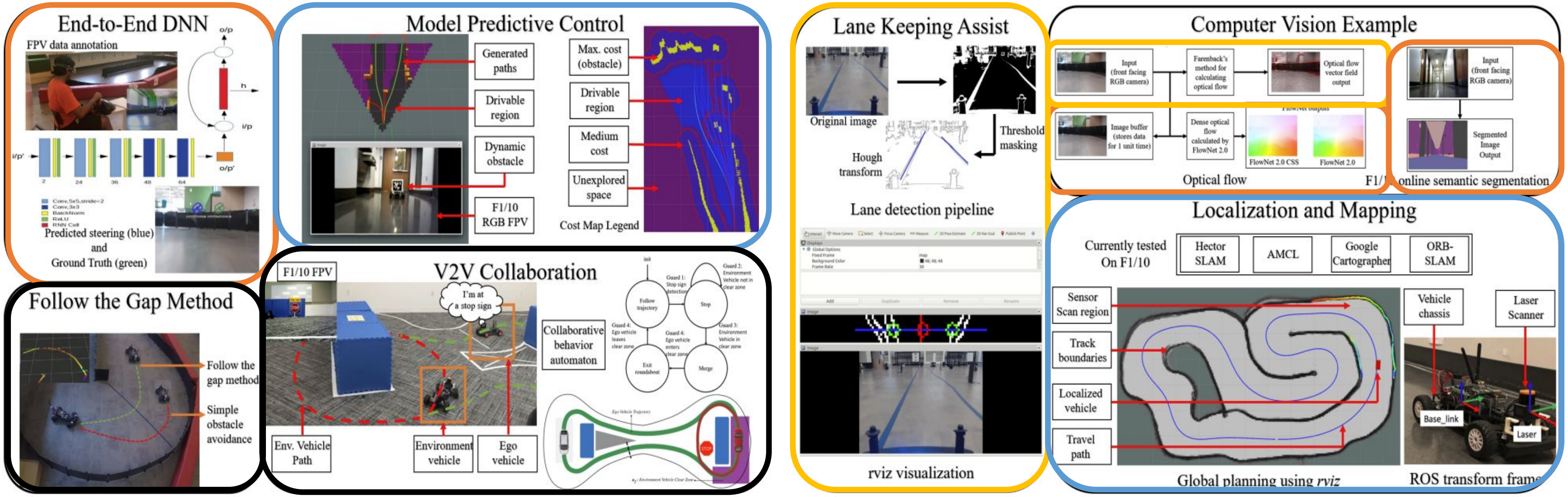
- Autonomous systems
  - Use deep neural networks (DNNs) for decisions.
  - Rely on continuous data-streams from sensors.
- System-on-Chips (SoCs)
  - Contain multiple domain-specific-accelerators (DSAs)
  - DSAs allow more efficient computation



Xavier NX: Common SoC onboard autonomous platforms

Autonomous System Example Workload - F1TENTH

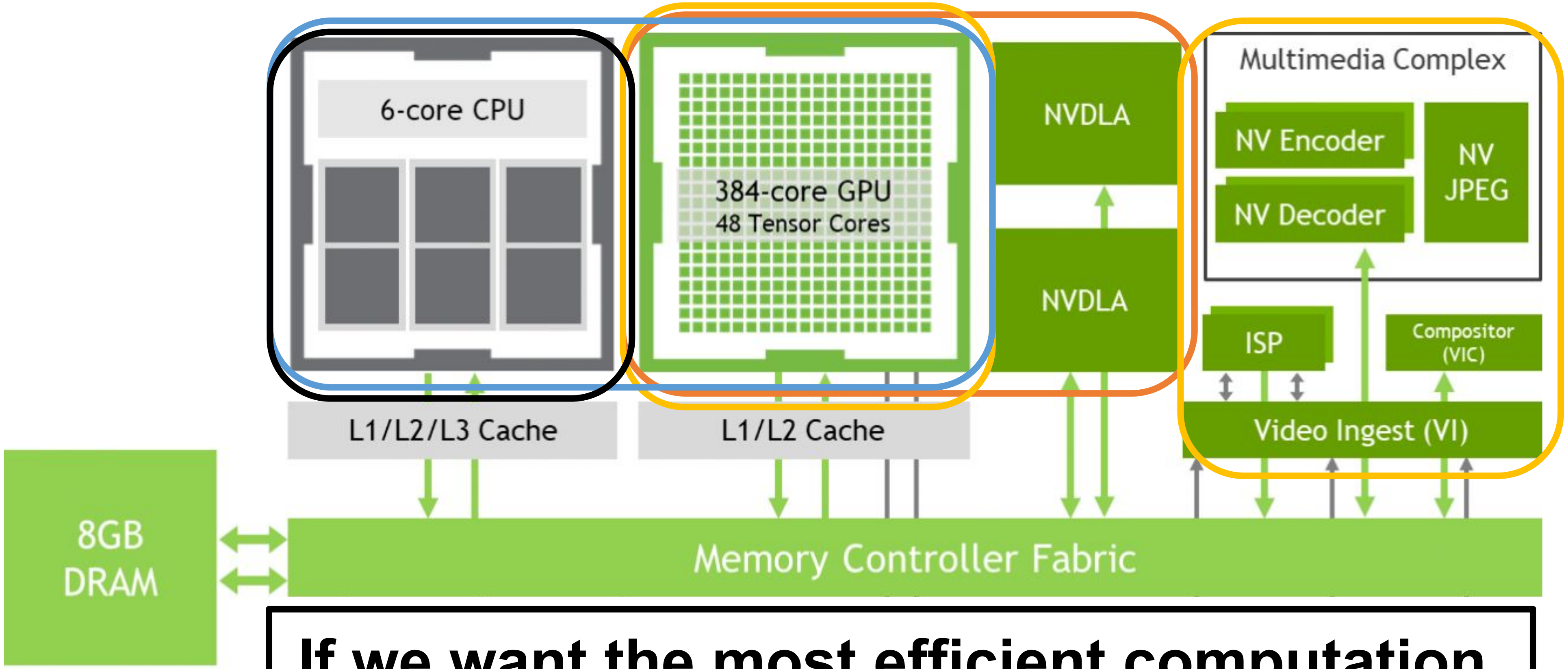
DNN Traditional CV Parallelizable CPU-Only



M. O'Kelly and V. Sukhil and H. Abbas and et al. F1/10: An Open-Source Autonomous Cyber-Physical Platform, 2019

Computation Onboard SoCs

1. DNNs
2. Traditional CV
3. Parallel Processors
4. General Processing

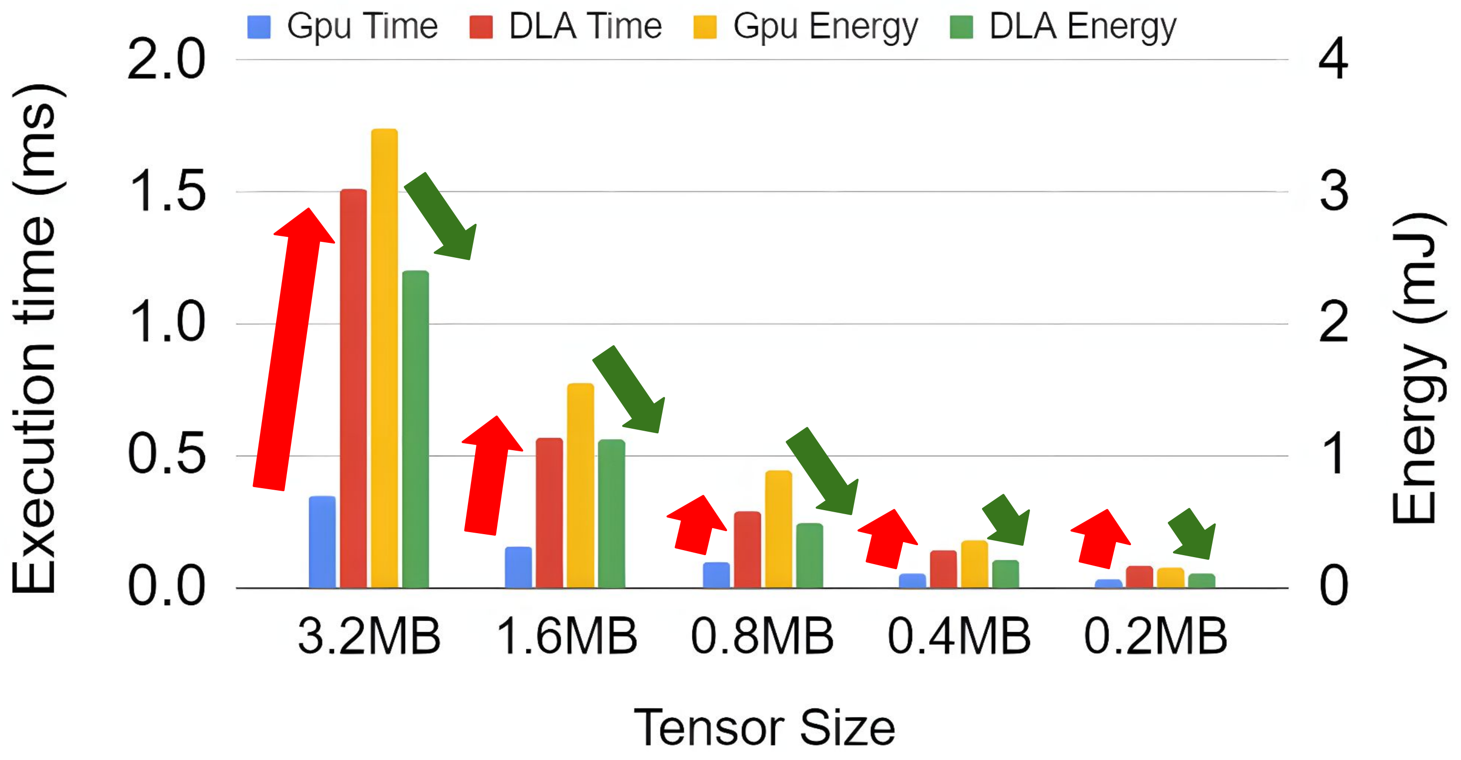


If we want the most efficient computation, we must use all available accelerators

https://developer.nvidia.com/blog/jetson-xavier-nx-the-worlds-smallest-ai-supercomputer/

Comparison of Accelerators

Observe an increase in overall latency



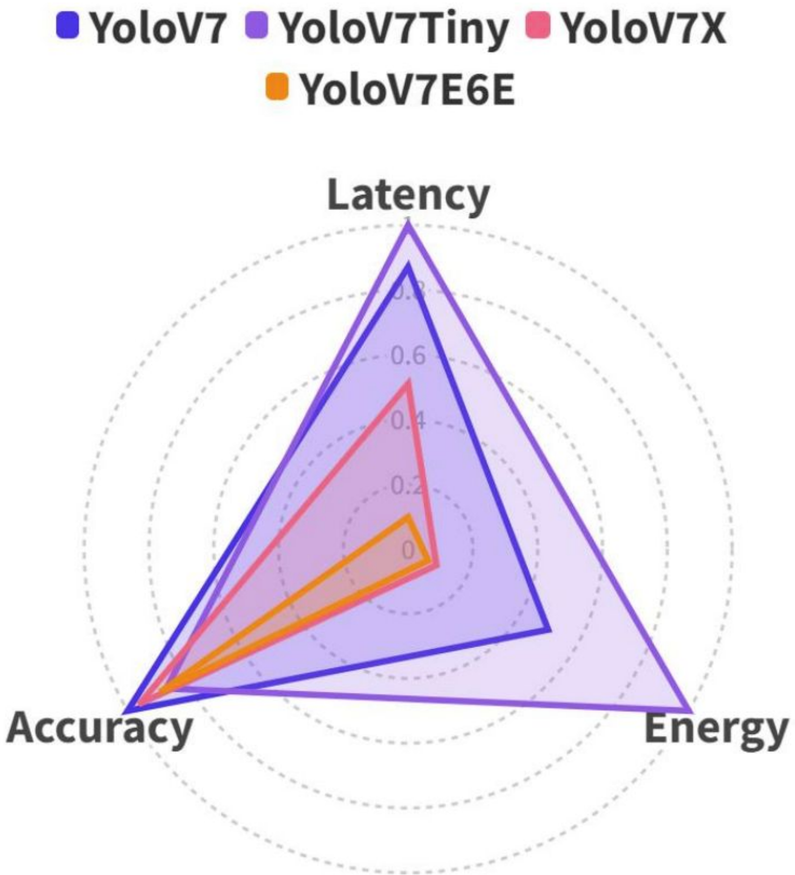
Observe a decrease in energy usage

Utilization of DSAs allows a different energy/latency tradeoff

Motivation

Object Detection on Autonomous SoCs

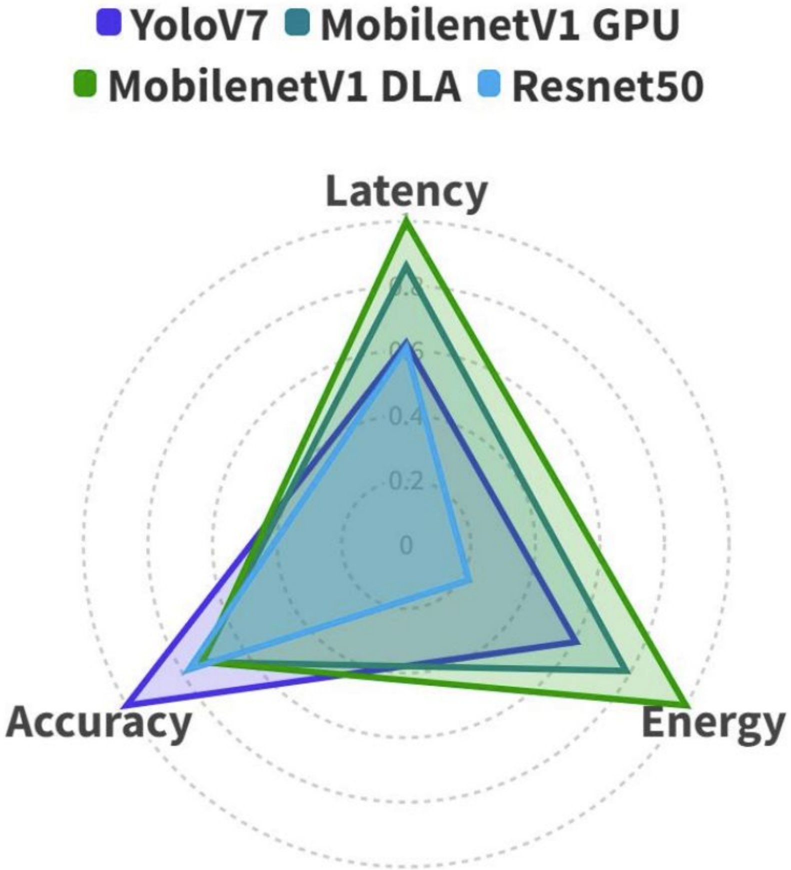
- Smaller/larger parameterizations
  - Allow accuracy/latency trade off between models
  - Larger models on edge platforms see increased latency and power draw
- Inter-Model Relationships
  - Strict monotonic relationships between energy, accuracy, and latency



Bigger is better

Object Detection using Multiple Models + Multiple Accelerators

- Running DNNs on multiple accelerators:
  - Adds scheduling complexity
  - Enables energy, accuracy, and latency tradeoffs
- Using multiple DNN architectures
  - Remove strict monotonic relationships

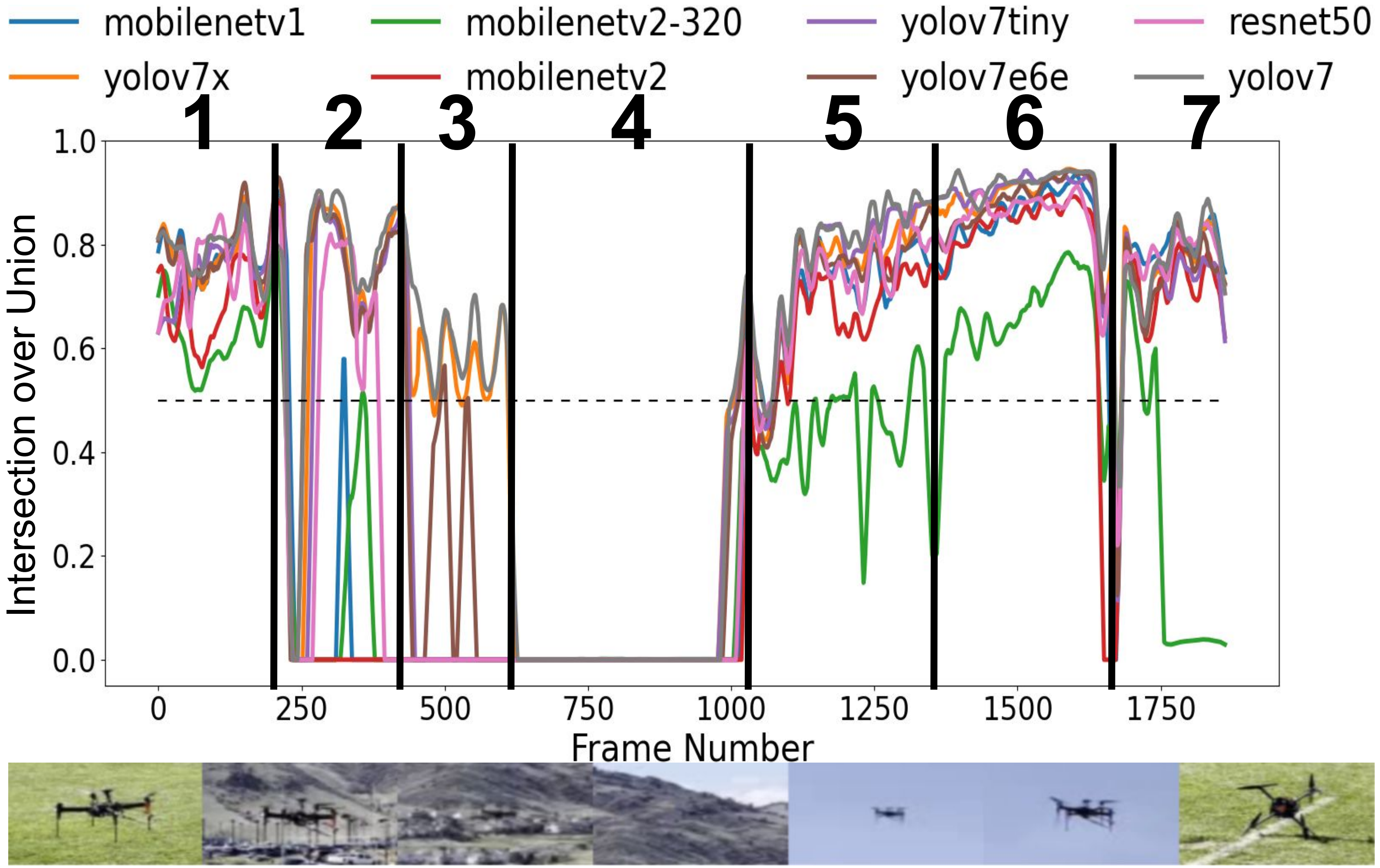


Bigger is better

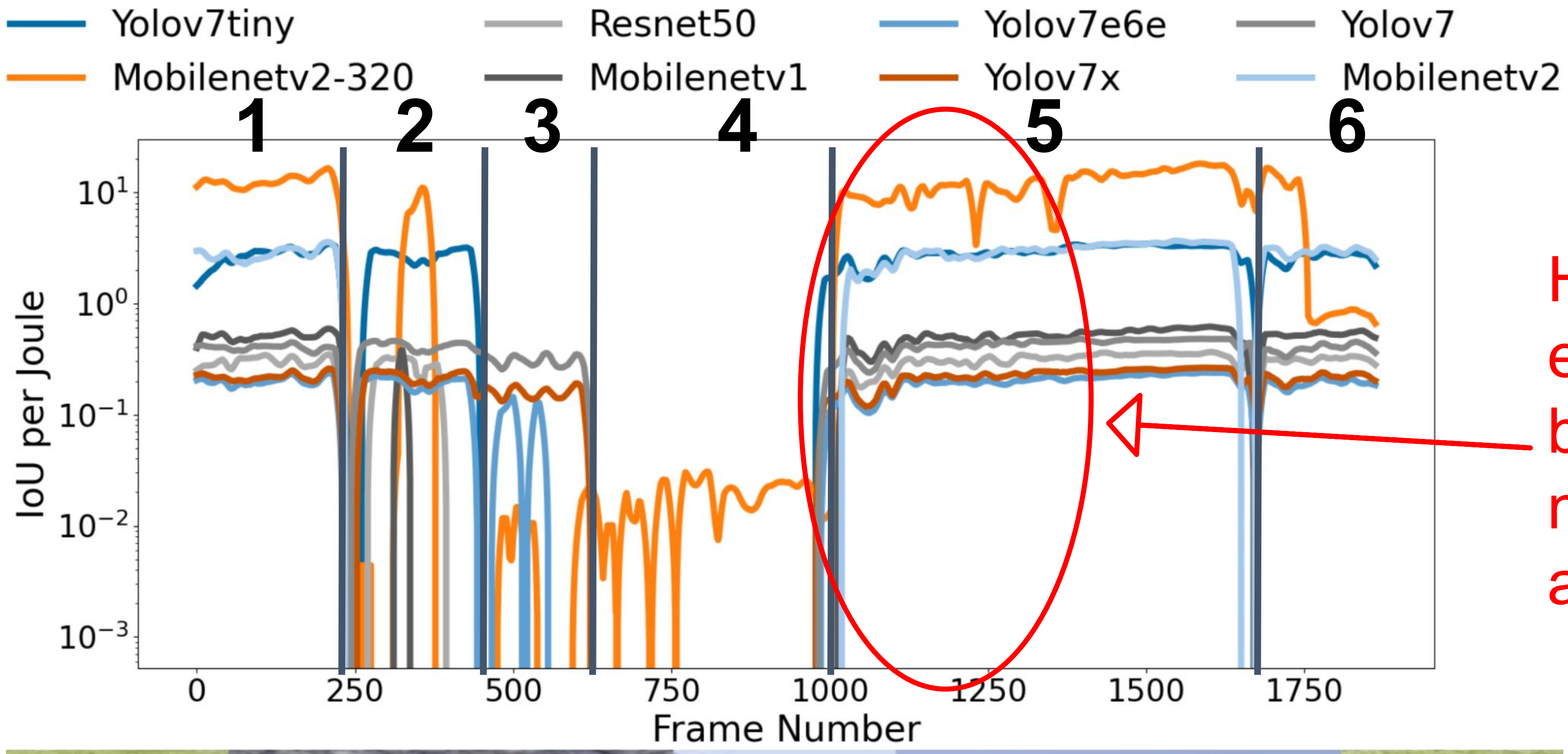
Multi-Model Inference

Which models achieve accuracy threshold:

1. All models
2. All YoloV7 + Resnet
3. YoloV7, YoloV7X
4. None
5. All except smallest
6. All models
7. All except smallest



Motivation - Multi-Model - Cont.

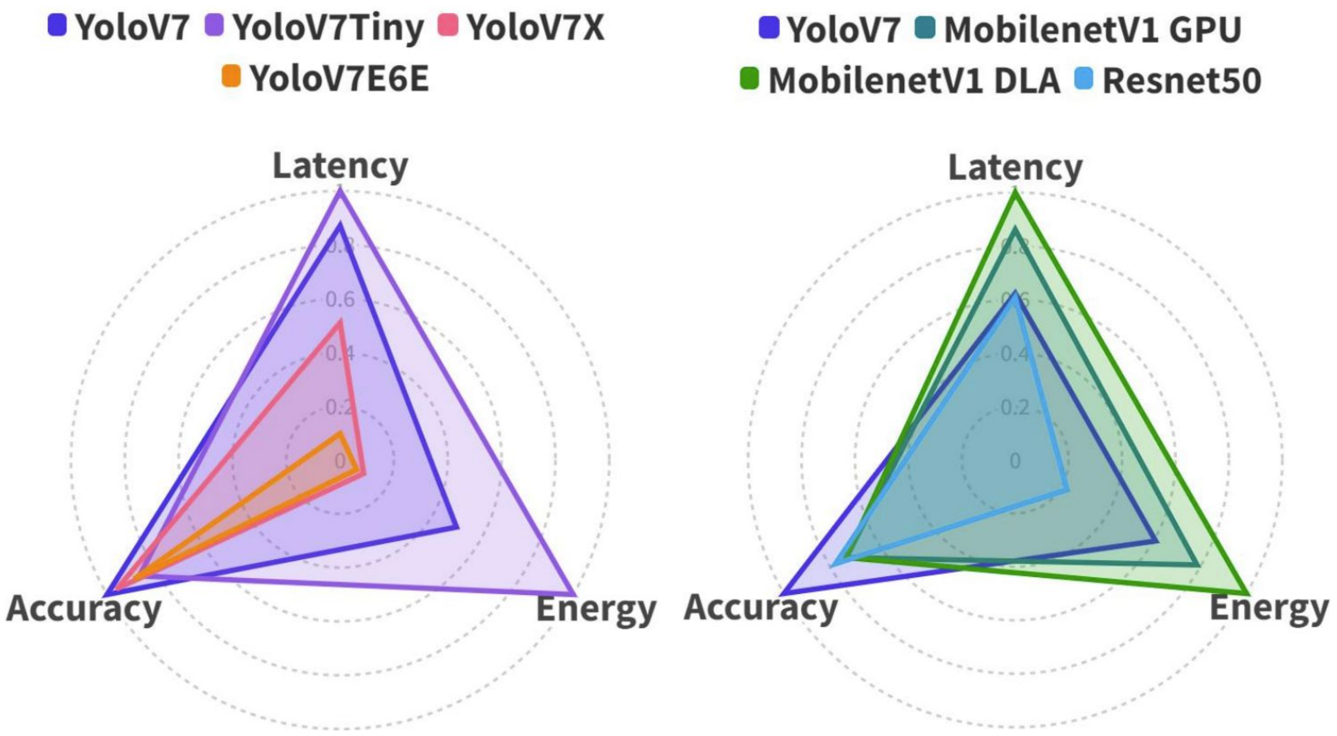


Highest efficiency but will not meet accuracy

Problem Statement

How can we utilize multiple models and multiple accelerators while optimizing for energy/latency?

- How do we know when we have chosen correctly?
- When do we switch between models or accelerators?

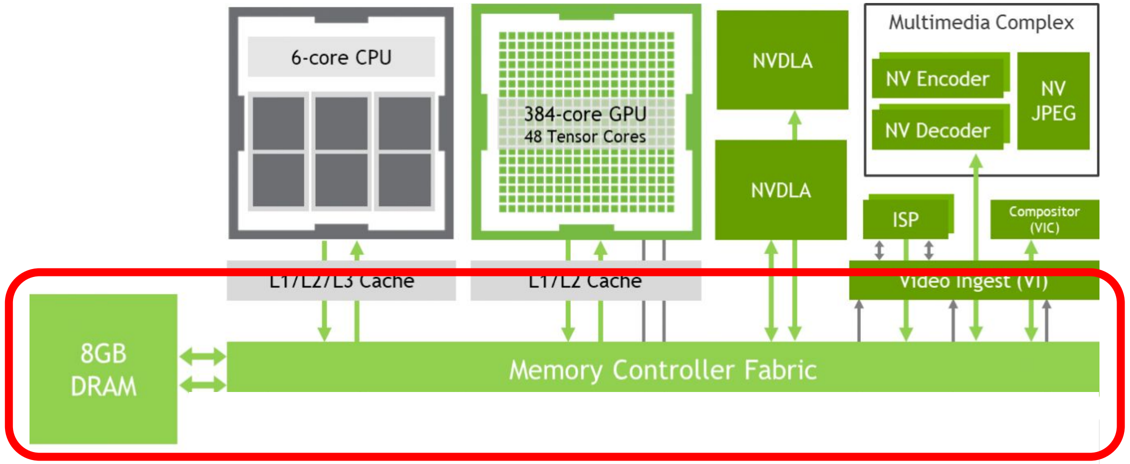


Challenges

1. We need to determine context at runtime.
2. We need to assess the current accuracy of models based purely on runtime context.
3. How many models we can load at once is restricted by the shared memory system.
4. We need to choose models without true knowledge of their prediction strength.

Where are we within our environment?

How does model X perform while the drone is here?



Shared memory limits individual capacity

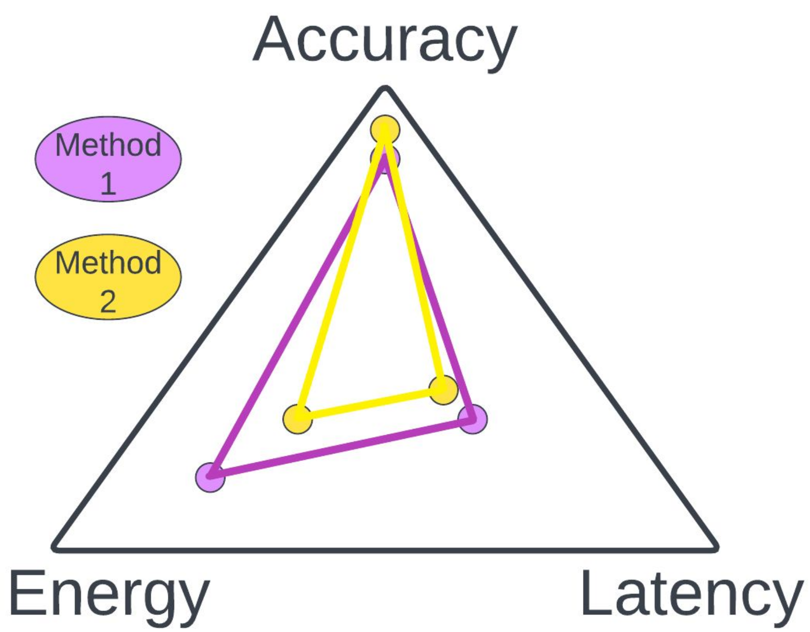
Related Work

Summary

Feature \ Related Work	Efficient edge object detection				Multi Model	Multi Accelerator		
	Glimpse [2]	MARLIN [5]	AdaVP [4]	RoaD-RuNNer [9]	Fast UQ [10]	Herald [11]	AxoNN [7]	***SHIFT***
Context Awareness	✗	✓	✓	✓	✗	✗	✗	✓
Multi-Accelerator	✗	✗	✗	✗	✗	✓	✓	✓
Multi-DNN	✗	✗	✗	✗	✓	✗	✗	✓
Energy-Aware	✗	✓	✓	✓	✗	✓	✓	✓
No-Offloading	✗	✓	✓	✗	✓	✓	✓	✓
Continuous	✓	✓	✗	✓	✗	✗	✗	✓

Related Work

- Continuous Detection
  - Offloading
  - Skipping frames
  - Efficiency optimizations
- DNN inference for multi-accelerators
  - Optimized schedules
  - Subgraphs
- Multi Model Detection
  - Multi-model scheme for pose prediction



Energy Efficient Detect-and-Track

- Pros:
  - Reduces energy usage by reducing number of object detection DNN inferences.
- Cons:
  - Adds an additional DNN inference for each frame
  - Processes asynchronously & skips frames

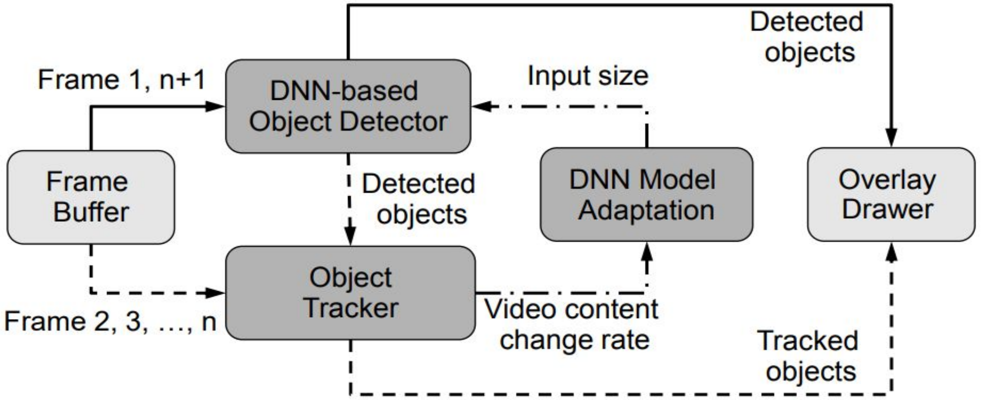


Fig. 3. Architecture of AdaVP. Each frame is either processed by the object detector or by the object tracker. The object tracker takes the objects detected by the object detector as input. The object detector uses the results of the object tracker to calculate the video content change rate and further adapt its DNN model setting. Finally, the processed frame will be passed to the overlay drawer module to draw the bounding boxes and display the frame on screen.

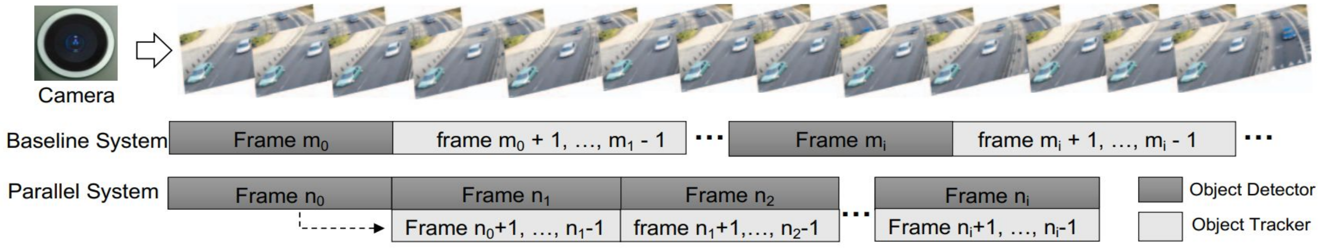
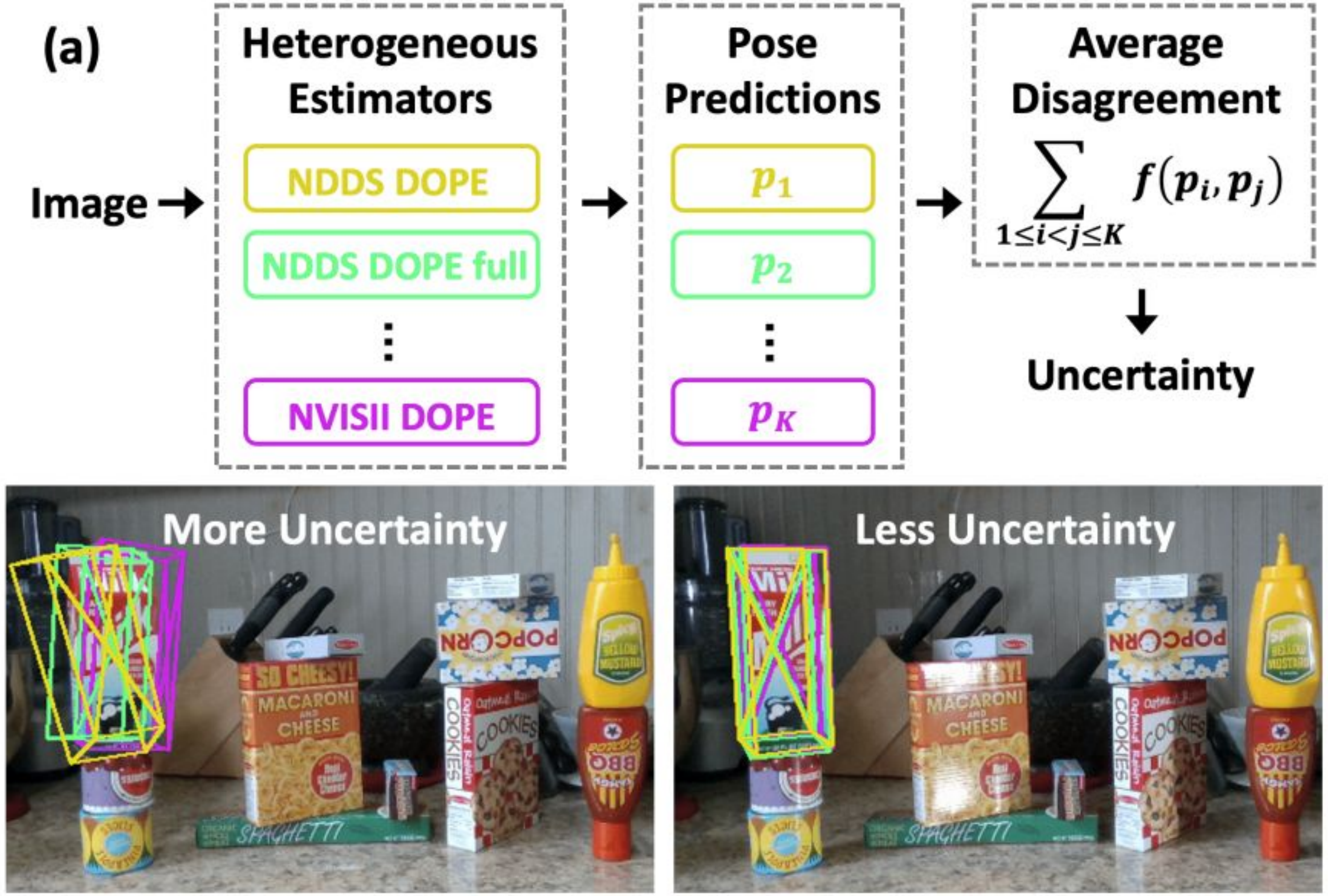


Fig. 4. Two different video processing systems, i.e., a baseline system and the pipeline of parallel detection and tracking.

M. Liu, X. Ding, and W. Du, “Continuous, real-time object detection on mobile devices without offloading,” in ICDCS’20

Multi-Model Inference

- Pros:
  - Multiple DNNs yield a better data spread compared to a single model
- Cons:
  - Need to perform multiple inferences per frame.

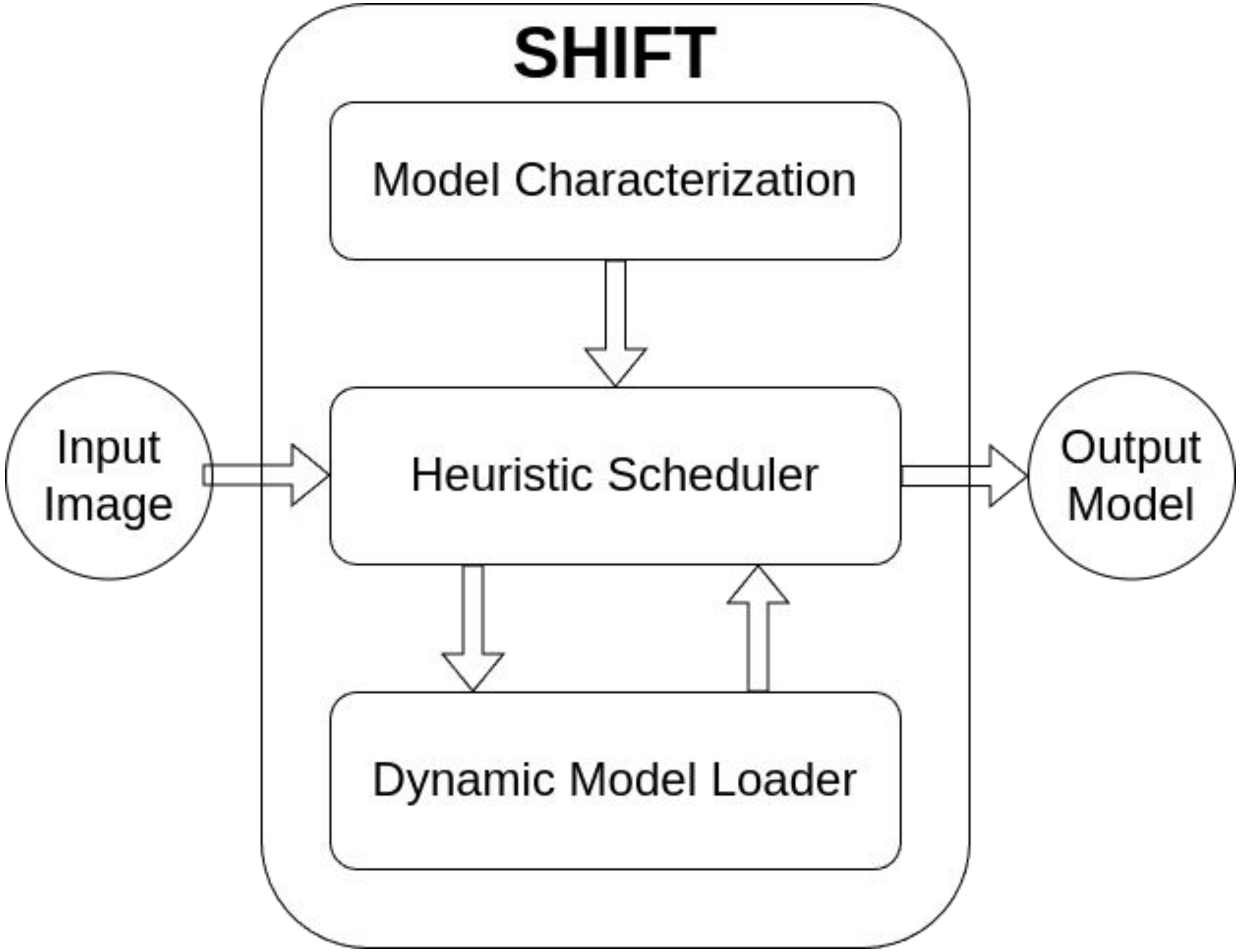


G. Shi, Y. Zhu, J. Tremblay, and et al., “Fast uncertainty quantification for deep object pose estimation,” in ICRA’21

Methodology

Overview of SHIFT

- Model Characterization
  - Identify key traits of each model
  - Construct confidence graph
- SHIFT Scheduler
  - Context detection
  - Heuristic scheduler
- Dynamic Model Loader
  - LRU model deallocation strategy



Model Characterization

Identified Model Traits
- Accuracy
- Confidence Score
- Latency
- Energy
- Model Loading Cost
  - Time
  - Memory
  - Energy

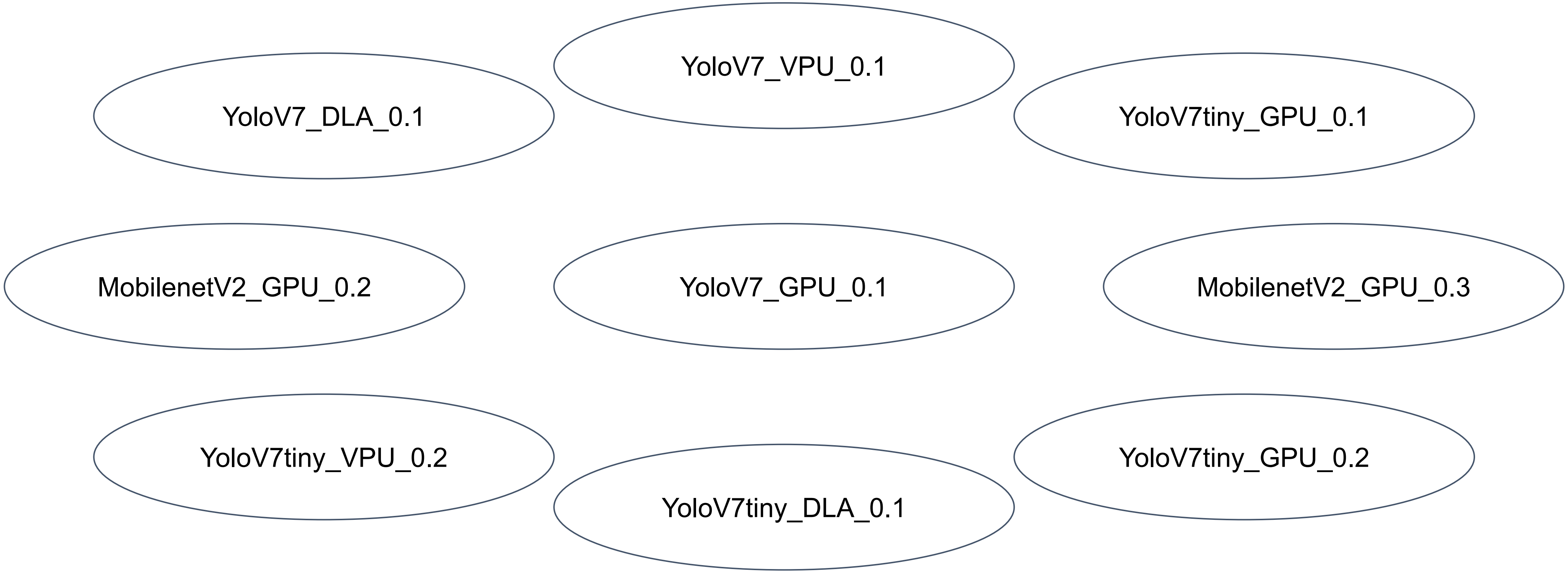
Prediction Methodology Goals
- Need to associate models offline without extra data
- Require fast predictions
- Deterministic model decisions
- Stable predictions

Confidence Graph

1. Create node for every model on every accelerator at every bin
2. Run every model on every accelerator on every image in validation set
3. Create edge weights from results of step 2
4. Process weights in neighborhood
    a. Neighborhood is defined as the one-hop adjacent nodes
5. Traverse with BFS
6. Aggregate common models for final predictions

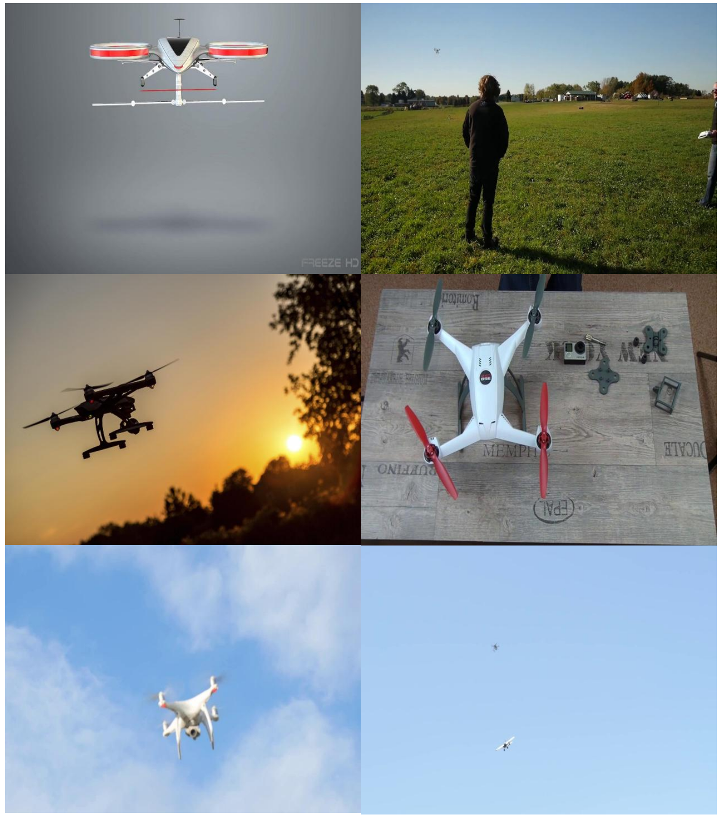
Yields ahead-of-time static predictions, O(1)

Confidence Graph - Nodes



Create a node in the graph for each model for each portion of the discrete confidence intervals

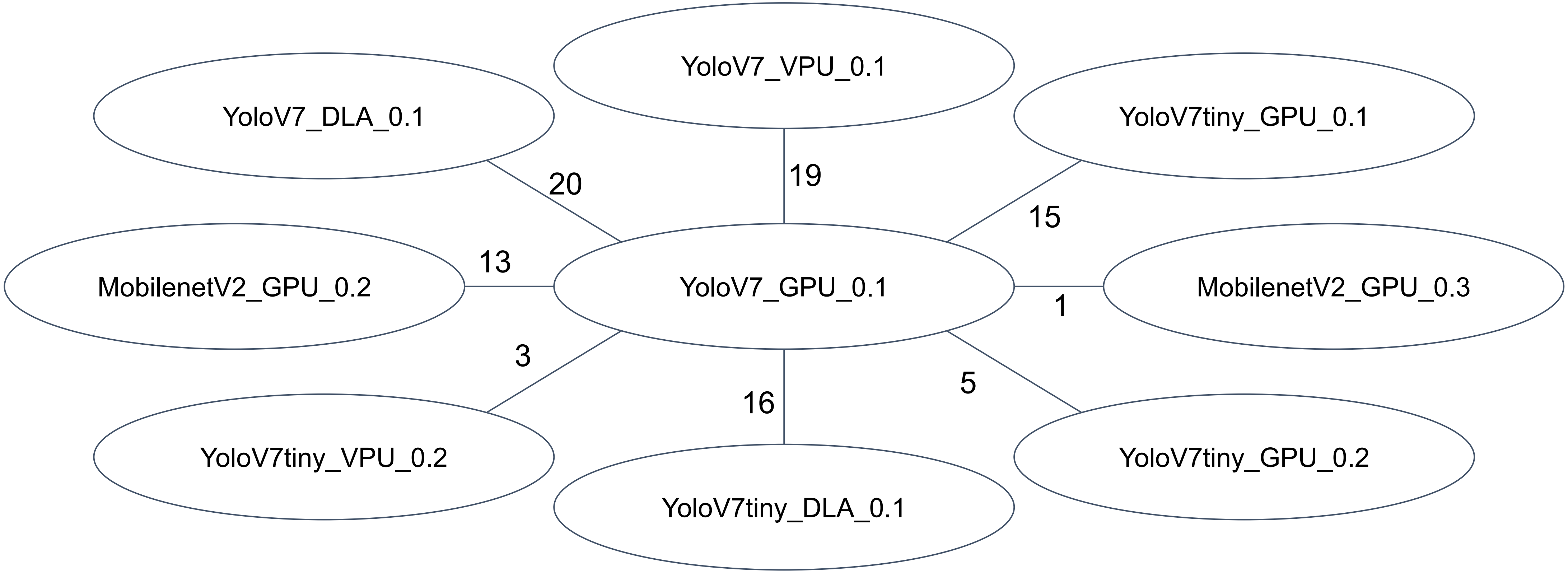
Confidence Graph - Performance



images	model1	model2	model3	model4
image1	0.1	0.0	0.19	0.36
image2	0.39	0.64	0.58	0.55
image3	0.63	0.46	0.84	0.48
image4	0.76	0.66	0.86	0.58
image5	0.81	0.76	0.97	0.57
image6	0.93	0.84	0.91	0.86
image7	0.75	0.95	0.52	0.91
image8	0.53	0.4	0.3	0.64
image9	0.75	0.55	0.72	0.83
image10	0.95	0.78	0.71	0.96
image11	0.92	0.71	0.79	0.64
image12	0.66	0.36	0.42	0.48
image13	0.82	0.73	0.63	0.7
image14	0.61	0.51	0.83	0.43
image15	0.62	0.74	0.35	0.75
image16	0.72	0.97	0.74	0.58
image17	0.61	0.58	0.84	0.44
image18	0.93	0.87	0.68	0.83
image19	0.5	0.38	0.79	0.35
image20	0.52	0.28	0.5	0.54

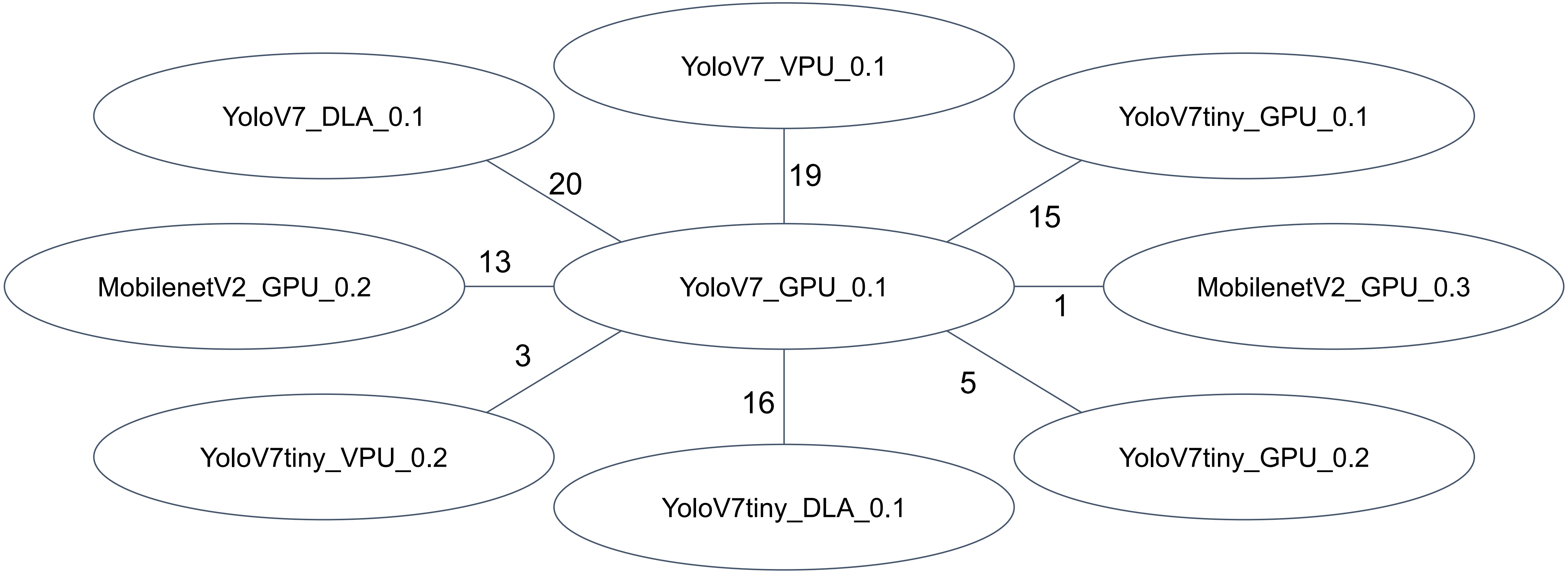
Run each available model on each accelerator on each image of the validation set

Confidence Graph - Edges



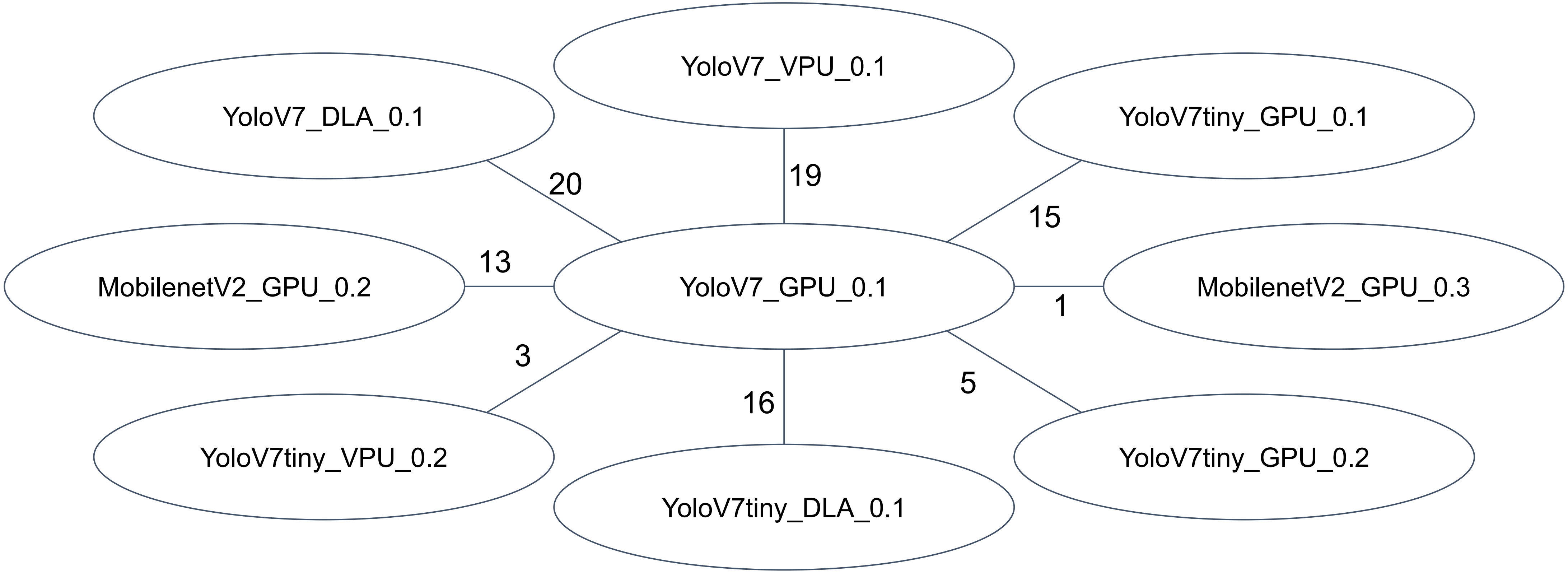
Increment the edge weight between two nodes if the model/confidence interval pairs are both present on the image

Confidence Graph - Weights



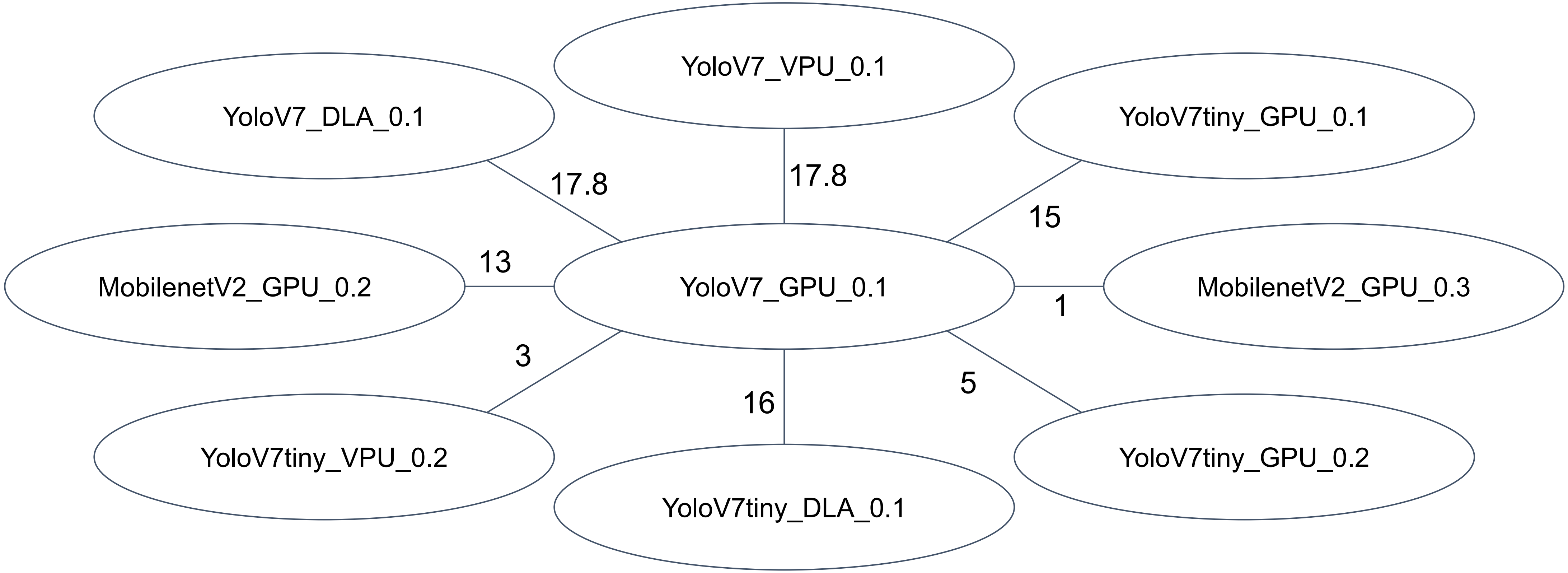
Clamp to a percentile, cull weak edges, normalize, and invert edge weights such that an edge weight has a lower is better standard

Confidence Graph - Weights - Clamp



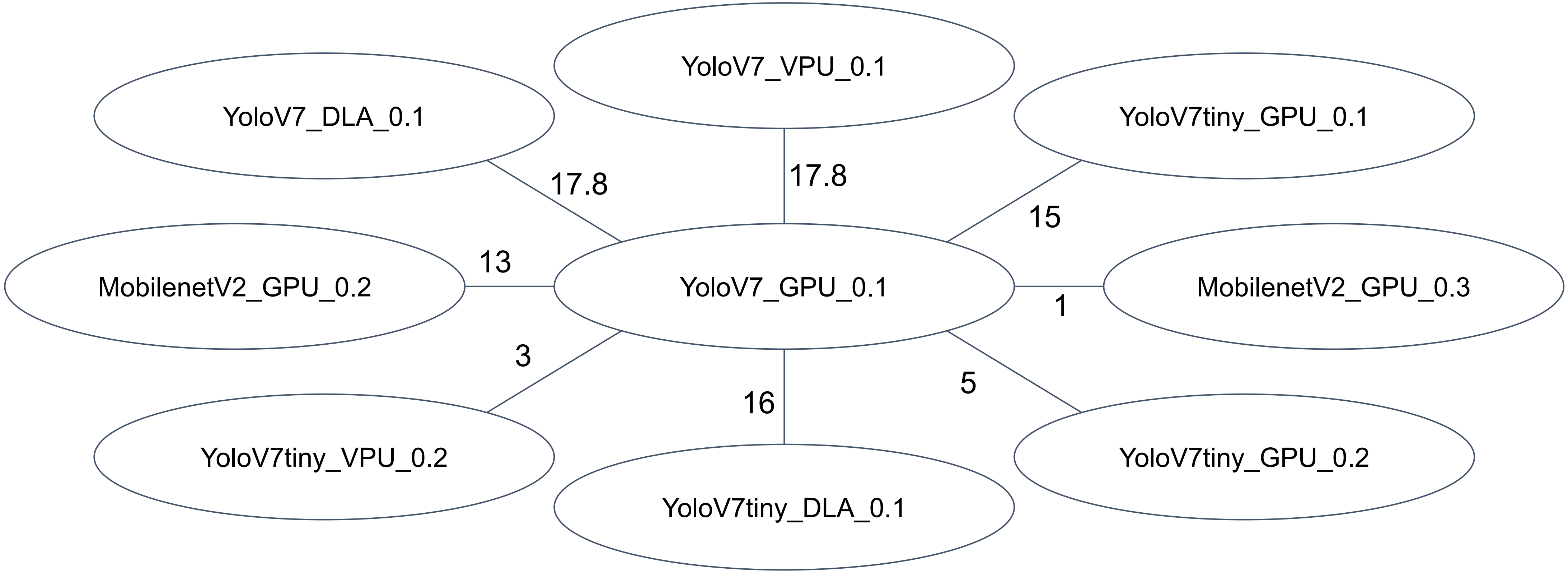
80th percentile of [1, 3, 5, 13, 15, 16, 19, 20] is 17.8

Confidence Graph - Weights - Clamp



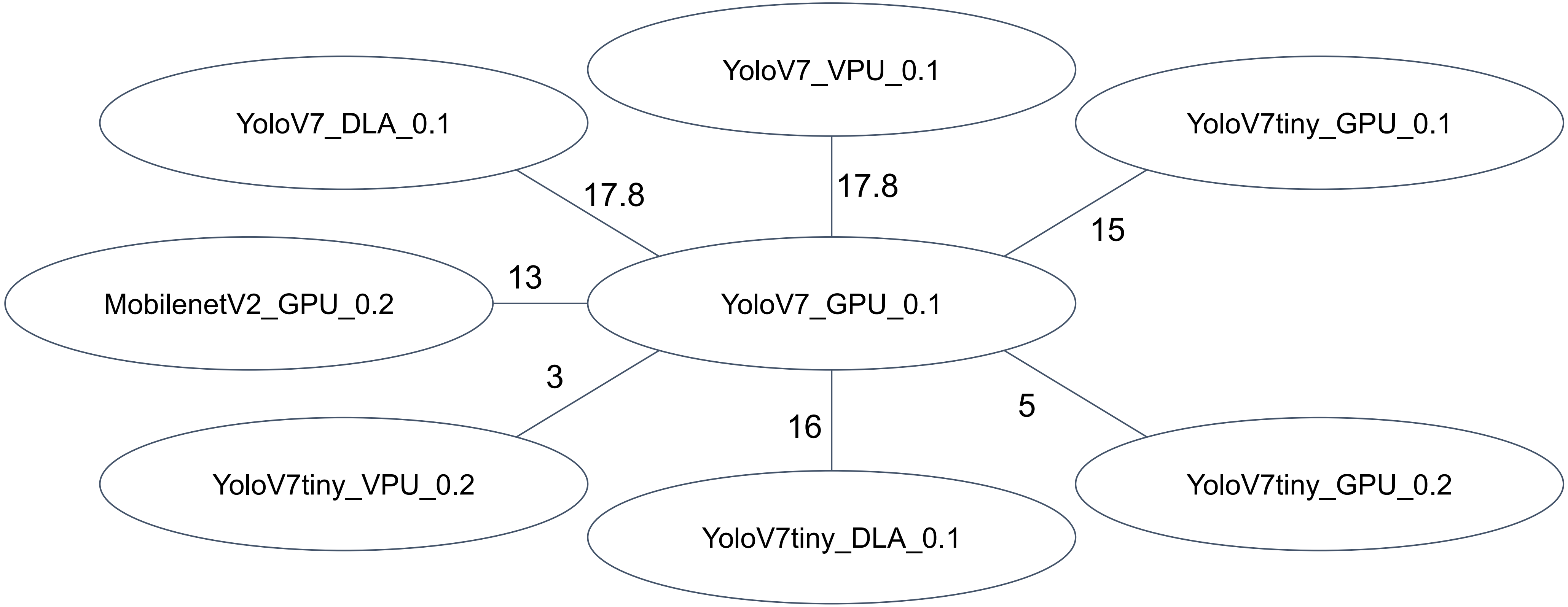
80th percentile of [1, 3, 5, 13, 15, 16, 19, 20] is 17.8

Confidence Graph - Weights - Cull



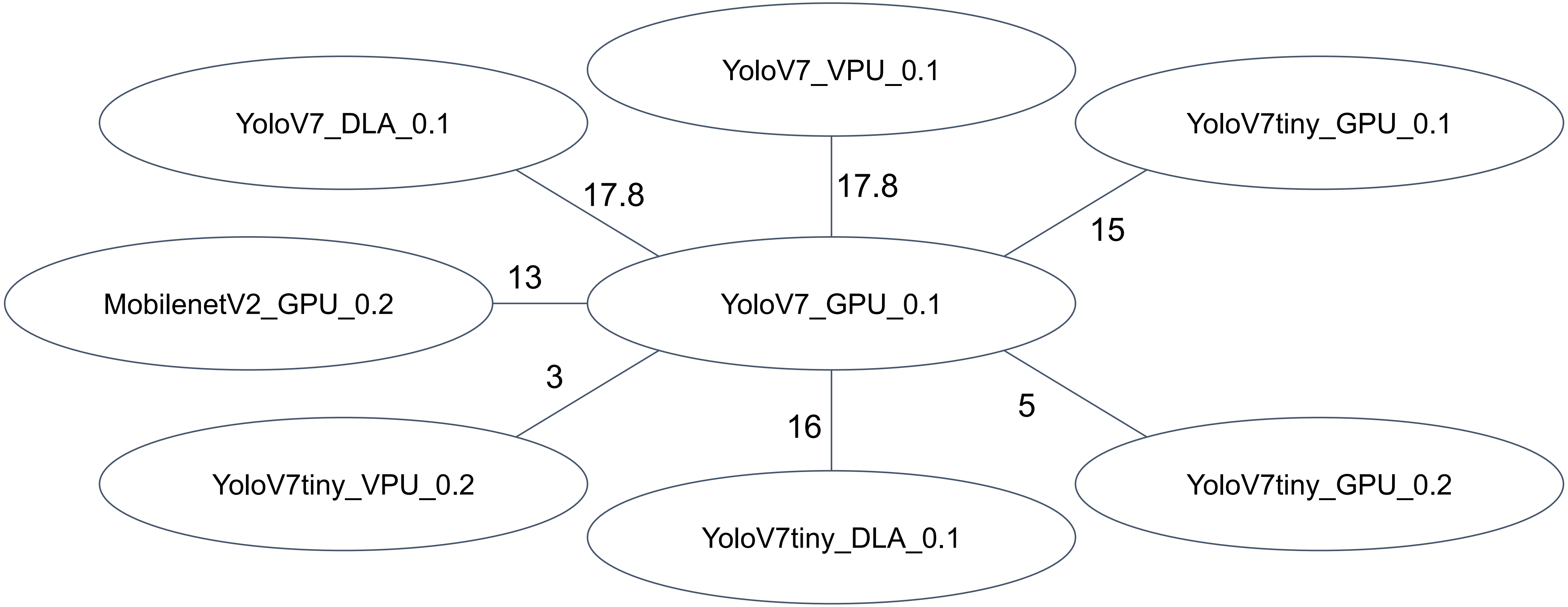
Edges with a single connection are too “noisy”

Confidence Graph - Weights - Cull



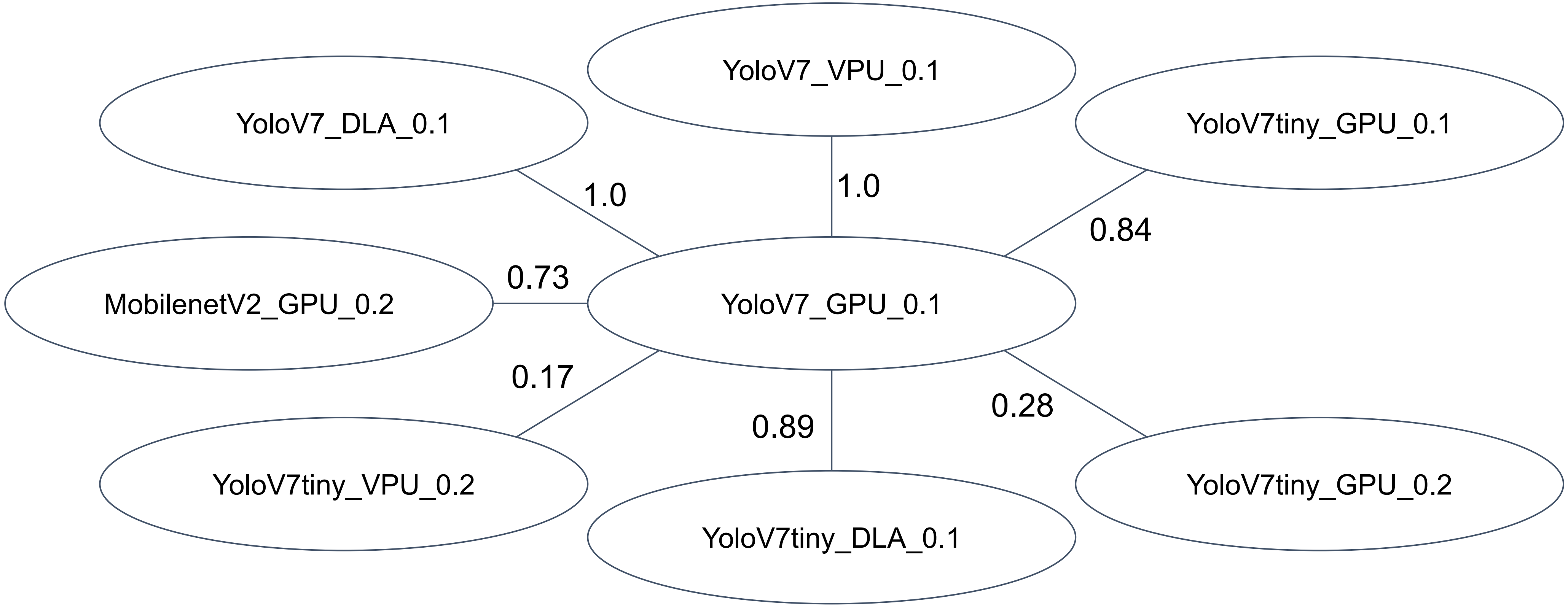
Edges with a single connection are too “noisy”

Confidence Graph - Weights - Normalize



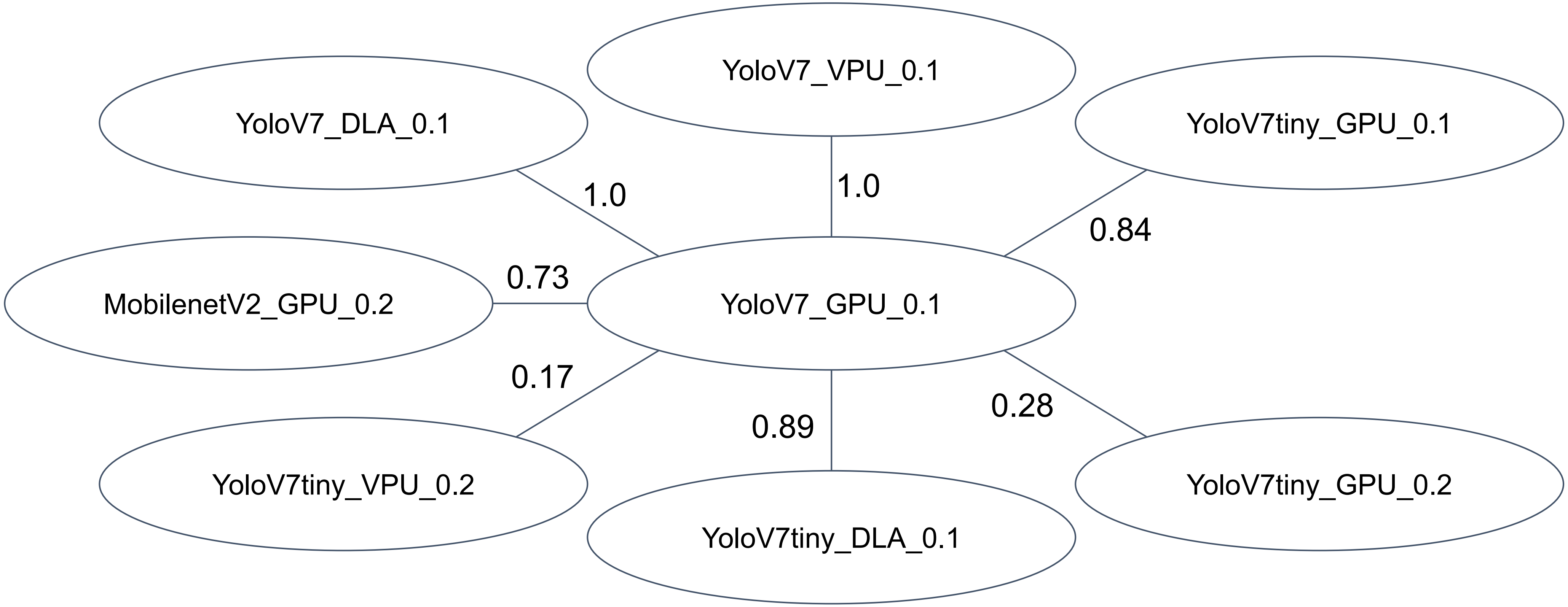
Divide all edge weights by the maximum weight in the neighborhood

Confidence Graph - Weights - Normalize



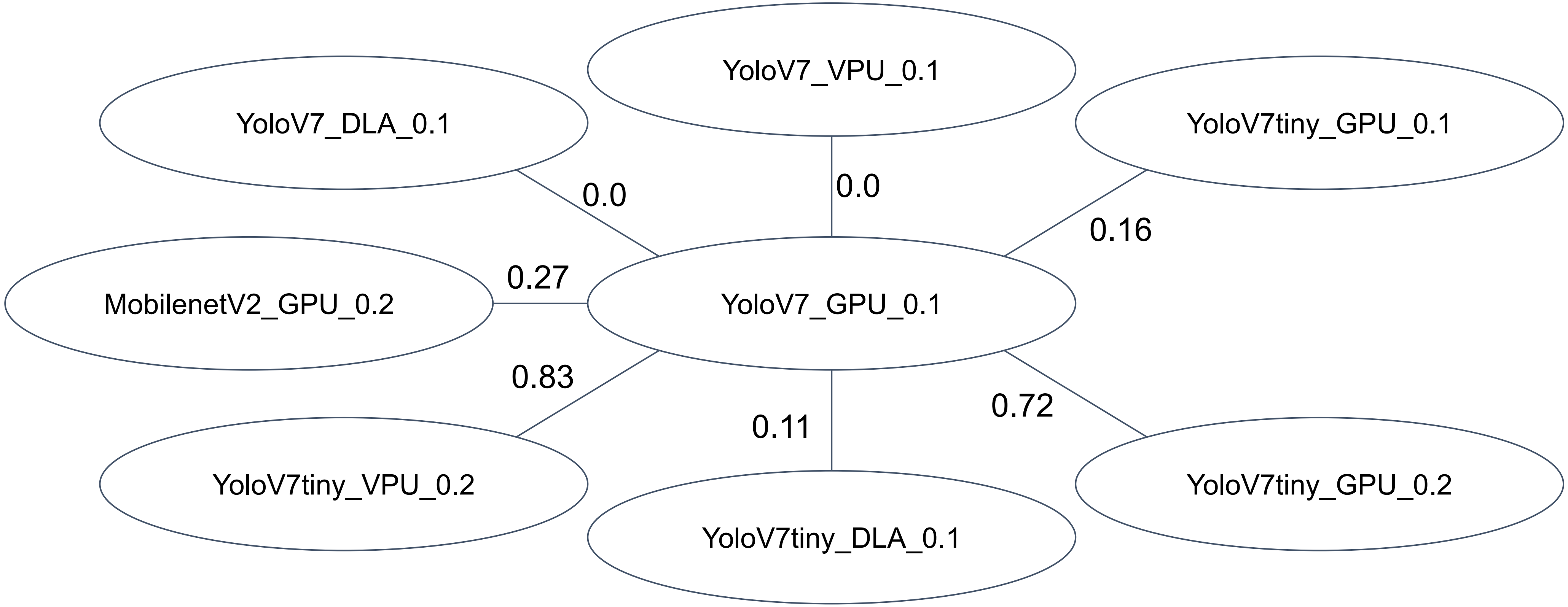
Divide all edge weights by the maximum weight in the neighborhood

Confidence Graph - Weights - Invert



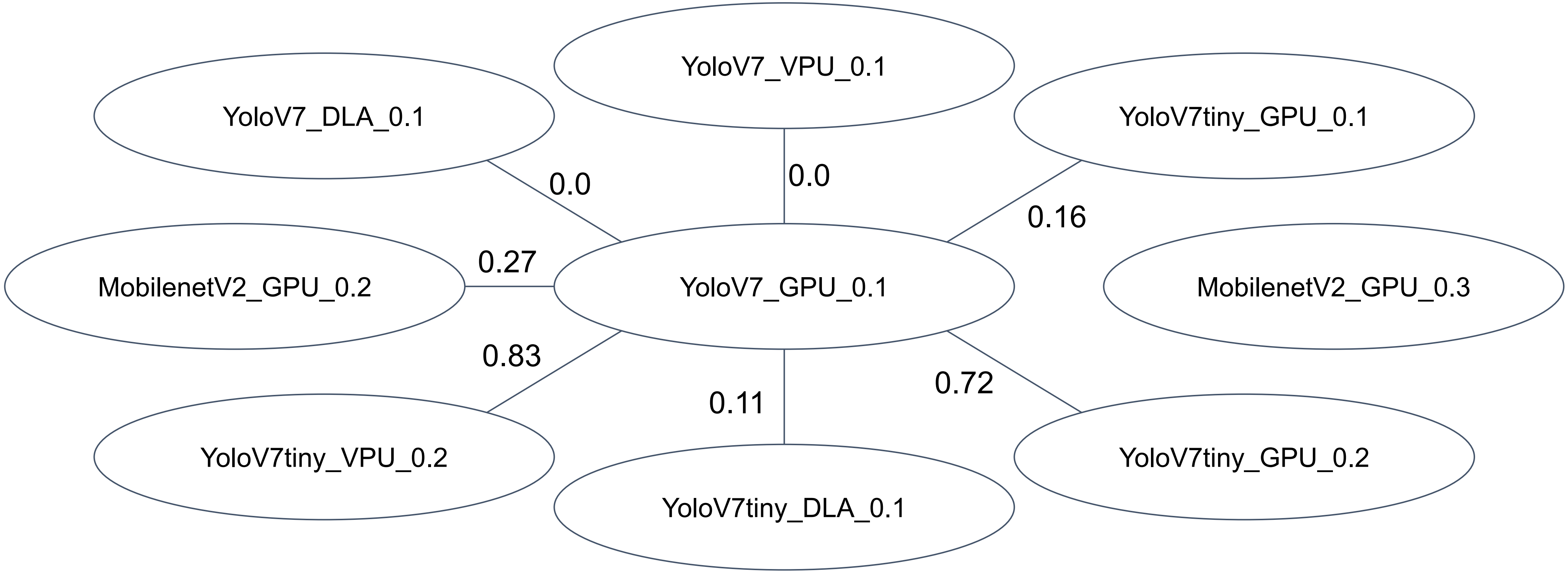
Invert edges by subtracting edge weight from 1.0

Confidence Graph - Weights - Invert



Invert edges by subtracting edge weight from 1.0

Confidence Graph - Final Edge Weights



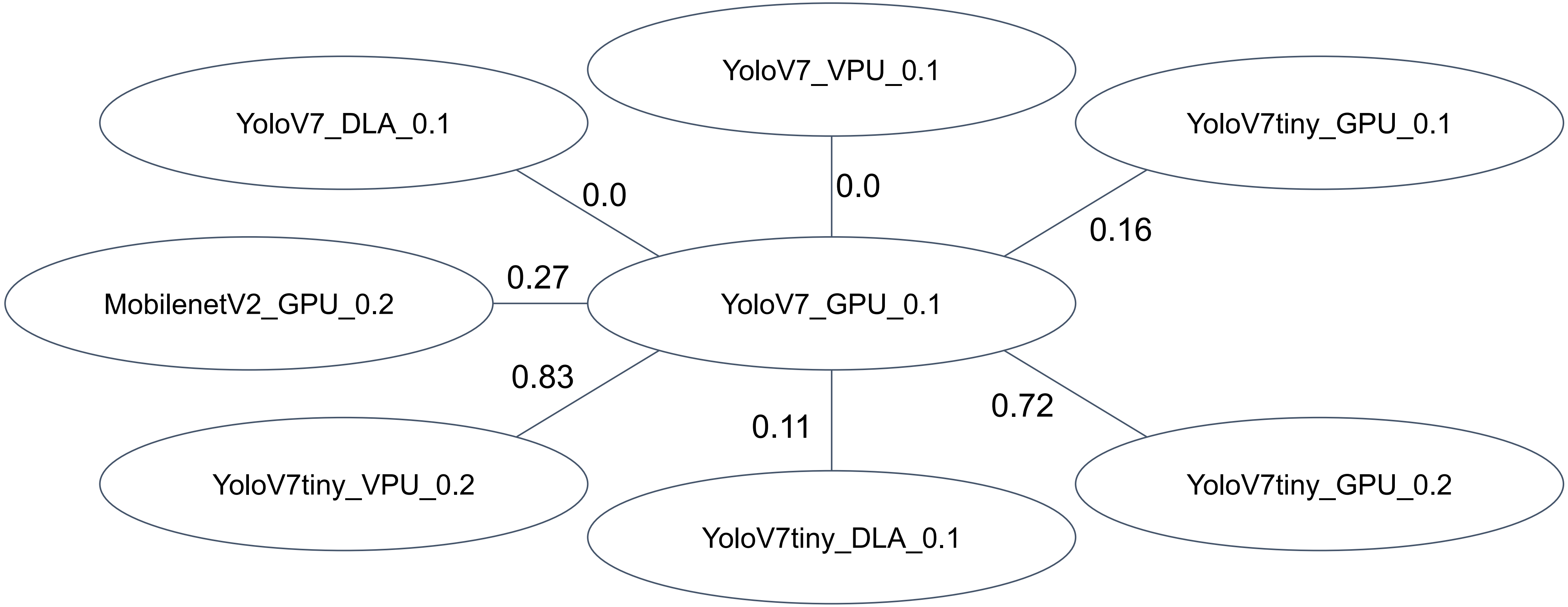
Final edge weights after the post processing stage. Post processing includes outlier removal, clamping, and inversion of weights.

Confidence Graph - BFS

An inference of YoloV7 on GPU with a confidence score between 0.1 and 0.2 yields:

Model	Accelerator	Accuracy	Cost	Number of Nodes
YoloV7	DLA	0.4	0.0	1
YoloV7	VPU	0.4	0.0	1
YoloV7 Tiny	GPU	0.3001	0.002	2
YoloV7 Tiny	DLA	0.3	0.73	1
MobilenetV2	GPU	0.3	0.41	1

Confidence Graph - BFS



Perform BFS traversal

Confidence Graph - BFS

YoloV7_GPU_0.1:

- YoloV7_DLA_0.1, 0.0
- YoloV7_VPU_0.1, 0.0
- YoloV7tiny_DLA_0.1, 0.11
- YoloV7tiny_GPU_0.1, 0.16
- MobilenetV2_GPU_0.2, 0.27
- YoloV7tiny_GPU_0.2, 0.72
- YoloV7tiny_VPU_0.2, 0.83

Perform BFS traversal

Confidence Graph - BFS

YoloV7_GPU_0.1:

- YoloV7_DLA_0.1, 0.0
- YoloV7_VPU_0.1, 0.0
- YoloV7tiny_DLA_0.1, 0.11
- YoloV7tiny_GPU_0.1, 0.16
- MobilenetV2_GPU_0.2, 0.27
- YoloV7tiny_GPU_0.2, 0.72
- YoloV7tiny_VPU_0.2, 0.83

Cut neighbors outside of the cost threshold, use 0.75 here

Confidence Graph - BFS

YoloV7_GPU_0.1:

- YoloV7_DLA_0.1, 0.0
- YoloV7_VPU_0.1, 0.0
- YoloV7tiny_DLA_0.1, 0.11
- YoloV7tiny_GPU_0.1, 0.16
- MobilenetV2_GPU_0.2, 0.27
- YoloV7tiny_GPU_0.2, 0.72

Cut neighbors outside of the cost threshold, use 0.75 here

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA_0.1, 0.0
- YoloV7_VPU_0.1, 0.0
- YoloV7tiny_DLA_0.1, 0.11
- YoloV7tiny_GPU_0.1, 0.16
- MobilenetV2_GPU_0.2, 0.27
- YoloV7tiny_GPU_0.2, 0.72

Aggregate model accuracies with a weighted average

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA_0.1, 0.0, 0.4
- YoloV7_VPU_0.1, 0.0, 0.4
- YoloV7tiny_GPU_0.1, 0.16, 0.3
- YoloV7tiny_GPU_0.2, 0.72, 0.35
- YoloV7tiny_DLA_0.1, 0.11, 0.3
- MobilenetV2_GPU_0.2, 0.27, 0.3

Aggregate model accuracies with a weighted average
Add the mean accuracy for the model in the range

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA_0.1, 0.0, 0.4
- YoloV7_VPU_0.1, 0.0, 0.4
- YoloV7tiny_GPU_0.1, 0.16, 0.3
- YoloV7tiny_GPU_0.2, 0.72, 0.35
- YoloV7tiny_DLA_0.1, 0.11, 0.3
- MobilenetV2_GPU_0.2, 0.27, 0.3

Compute a weighted average, first transform weights using:
w = max(((cost_threshold - w) / cost_threshold) ** 2, 1e-8)

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA: [(1.0, 0.4)]
- YoloV7_VPU: [(1.0, 0.4)]
- YoloV7tiny_GPU: [(0.62, 0.3), (0.001, 0.35)]
- YoloV7tiny_DLA: [(0.73, 0.3)]
- MobilenetV2_GPU: [(0.41, 0.3)]

Compute a weighted average, first transform weights using:
w = max(((cost_threshold - w) / cost_threshold) ** 2, 1e-8)

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA: [(1.0, 0.4)]
- YoloV7_VPU: [(1.0, 0.4)]
- YoloV7tiny_GPU: [(0.62, 0.3), (0.001, 0.35)]
- YoloV7tiny_DLA: [(0.73, 0.3)]
- MobilenetV2_GPU: [(0.41, 0.3)]

Compute the estimated accuracy with a weighted average

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA: acc: 0.4
- YoloV7_VPU: acc: 0.4
- YoloV7tiny_GPU: acc: 0.30008
- YoloV7tiny_DLA: acc: 0.3
- MobilenetV2_GPU: acc: 0.3

Compute the estimated accuracy with a weighted average

Confidence Graph - BFS - Aggregate

YoloV7_GPU_0.1:

- YoloV7_DLA: acc: 0.4
- YoloV7_VPU: acc: 0.4
- YoloV7tiny_GPU: acc: 0.30008
- YoloV7tiny_DLA: acc: 0.3
- MobilenetV2_GPU: acc: 0.3

Now we have the static accuracy of all models on all accelerators given one model on an accelerator with a certain confidence score. These predictions can be pre-computed and stored in a hashmap for O(1) accuracy predictions.

SHIFT Scheduler

- Set of energy/accuracy/latency knobs for user adjustment
- Computes image similarity across entire image and detected bounding boxes to determine if context is changing
- Minimizes perceived cost

$$\text{NCC}(p, c) = \frac{\sum (p - mean(p))(c - mean(c))}{(\sqrt{\sum (c - mean(c))^2} \times \sqrt{\sum (p - mean(p))^2}} \quad (1)$$

Algorithm 1 Model Scheduling

```
procedure SHIFT SCHEDULE(m, c, i, b)
    s = min(NCC(lastImage, i), NCC(lastBbox, b))
    if s × c ≥ accuracyThreshold then
        return m
    end if
    E = scheduler.energy                          ▷ 0 → 1 model energy
    L = scheduler.latency                         ▷ 0 → 1 model latency
    W = scheduler.weights                         ▷ Tuned knobs
    C = graphPredict(m, c)                        ▷ set of (name, acc, dist)
    R, scores = map(), map()
    for (n, a, d) ∈ C do
        a.Buffer.append(a)
        R[n] = average(a.Buffer)
    end for
    V = {n | n ∈ R, n ≥ accuracyThreshold}
    if length(V) == 0 then
        V = R
    end if
    for n ∈ R.keys() do
        s = R[n] * W[0] + E[n] * W[1] + L[n] * W[2]
        scores[n] = s
    end for
    return max(scores)
end procedure
```

Dynamic Model Loader

- Models occupy memory
- SoCs utilize shared memory (commonly)
- Too many models allocated will lead to programs being killed
- Deallocate models using LRU to save memory

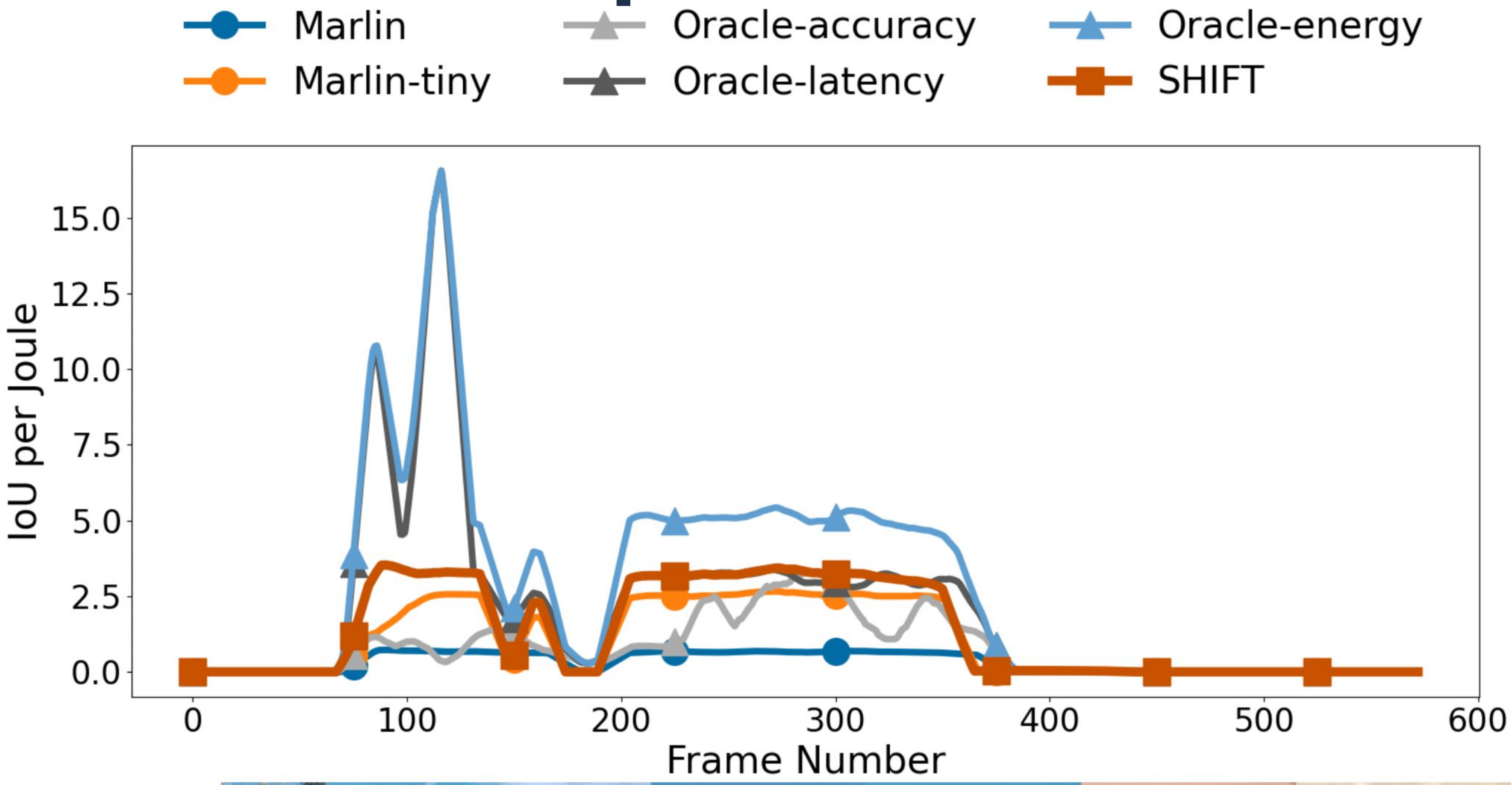
Results

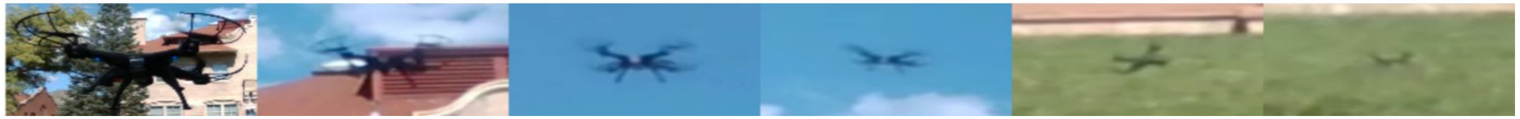
Results - All Models

Model Name	Accuracy		Avg. Time (s)			Avg. Energy (Joules)			Avg. Power Draw (W)		
	Avg. IoU	Success Rate	GPU	GPU/DLA	OAK-D	GPU	GPU/DLA	OAK-D	GPU	GPU/DLA	OAK-D
YoloV7-E6E	0.564	65.8%	0.255	0.221	-	3.947	1.228	-	15.48	5.56	-
YoloV7-X	0.593	71.1%	0.222	0.195	-	3.586	1.088	-	16.15	5.57	-
YoloV7	0.618	74.1%	0.130	0.118	0.894	1.968	0.656	1.391	15.14	5.56	1.56
YoloV7-Tiny	0.533	64.0%	0.025	0.024	0.107	0.280	0.134	0.206	11.2	5.58	1.93
SSD Resnet50	0.480	58.9%	0.151	0.138	-	2.504	0.816	-	16.58	5.91	-
SSD MobilenetV1	0.452	55.4%	0.094	0.092	-	1.519	0.561	-	16.16	6.10	-
SSD MobilenetV2	0.401	51.3%	0.023	0.058	-	0.248	0.307	-	10.78	5.29	-
SSD MobilenetV2 320x320	0.304	36.2%	0.009	0.023	-	0.046	0.100	-	5.11	4.35	-

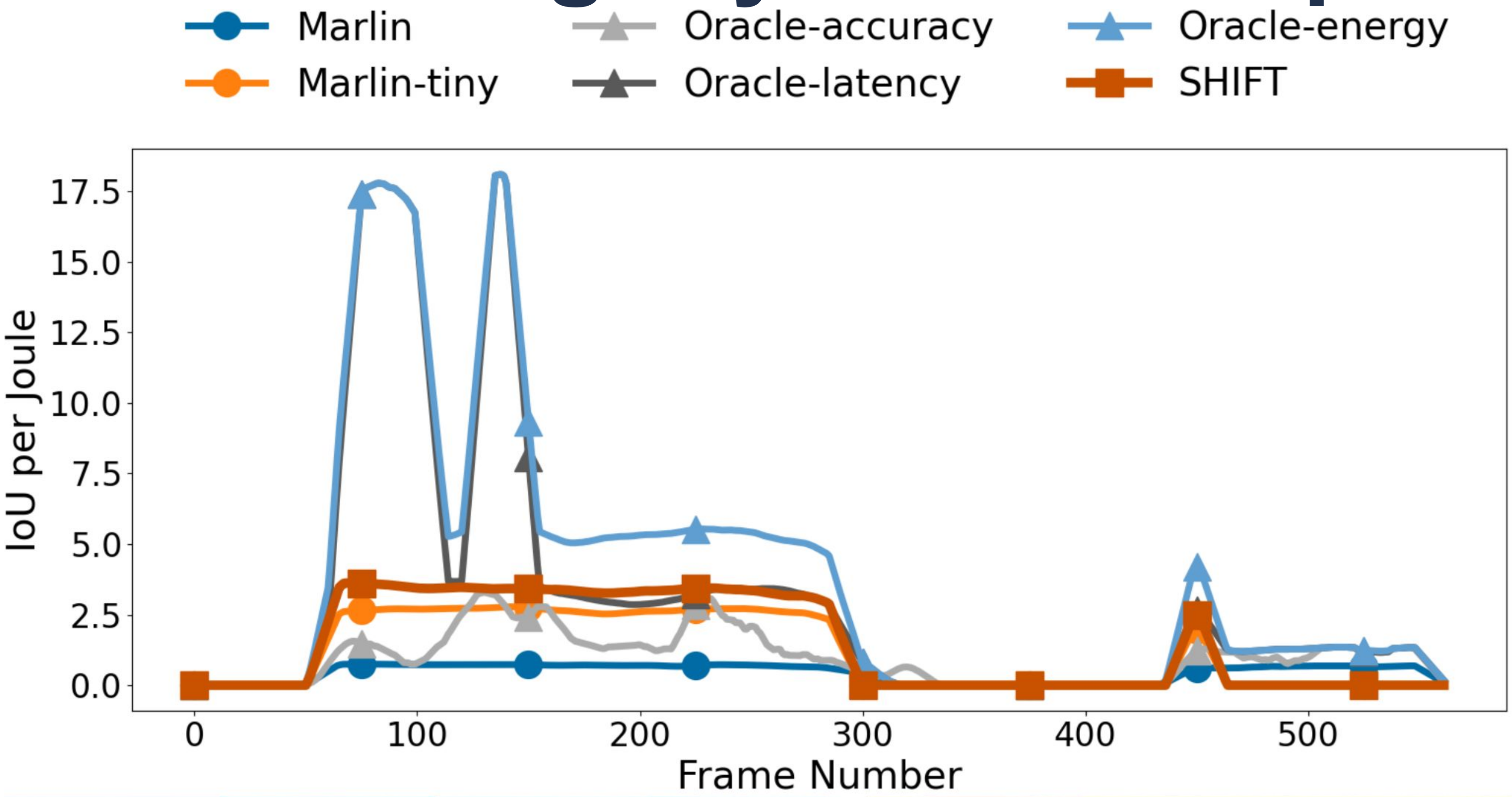
YoloV7 on GPU achieves highest success rate across our tests. This model serves as our baseline

Scenario 1 - Simple

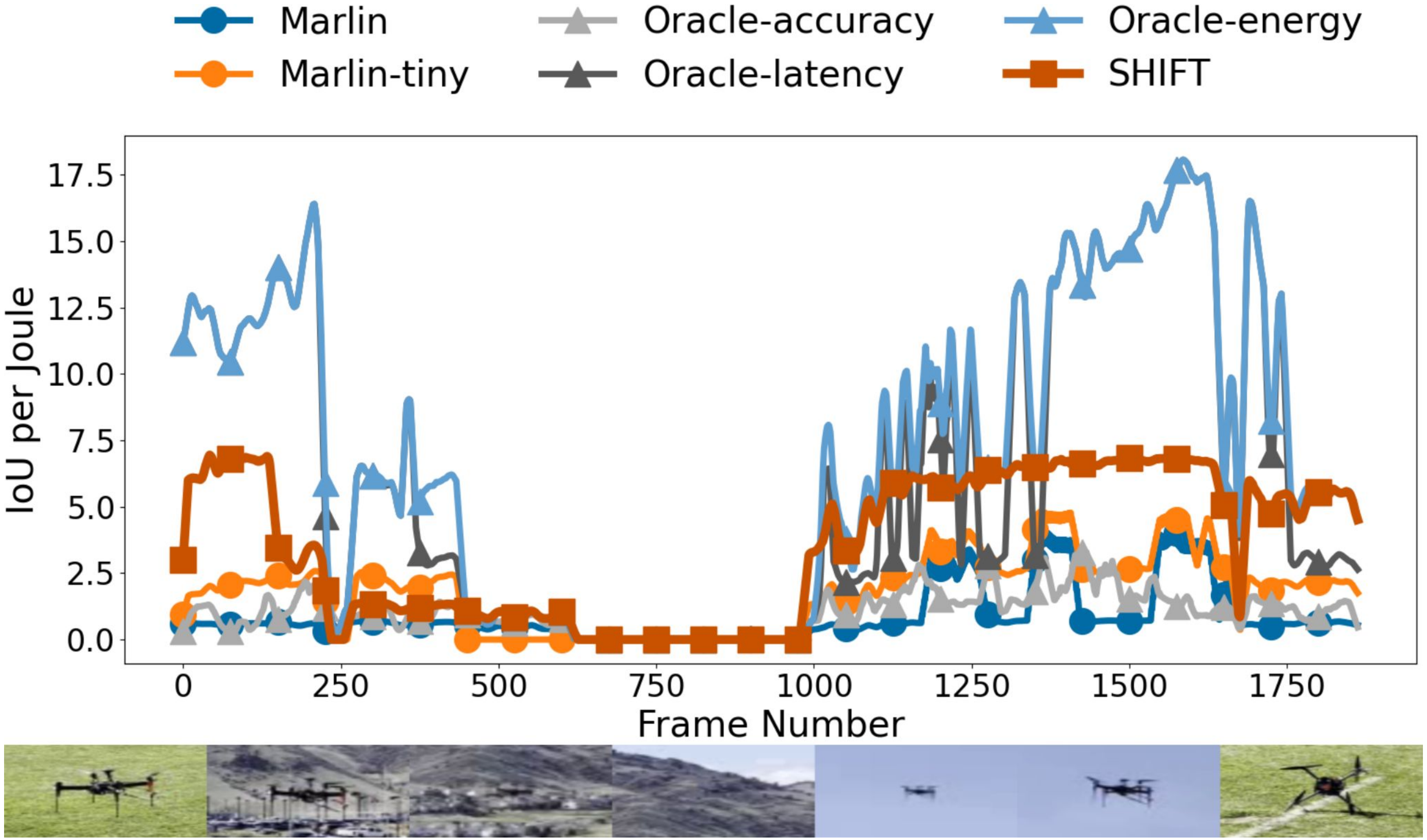




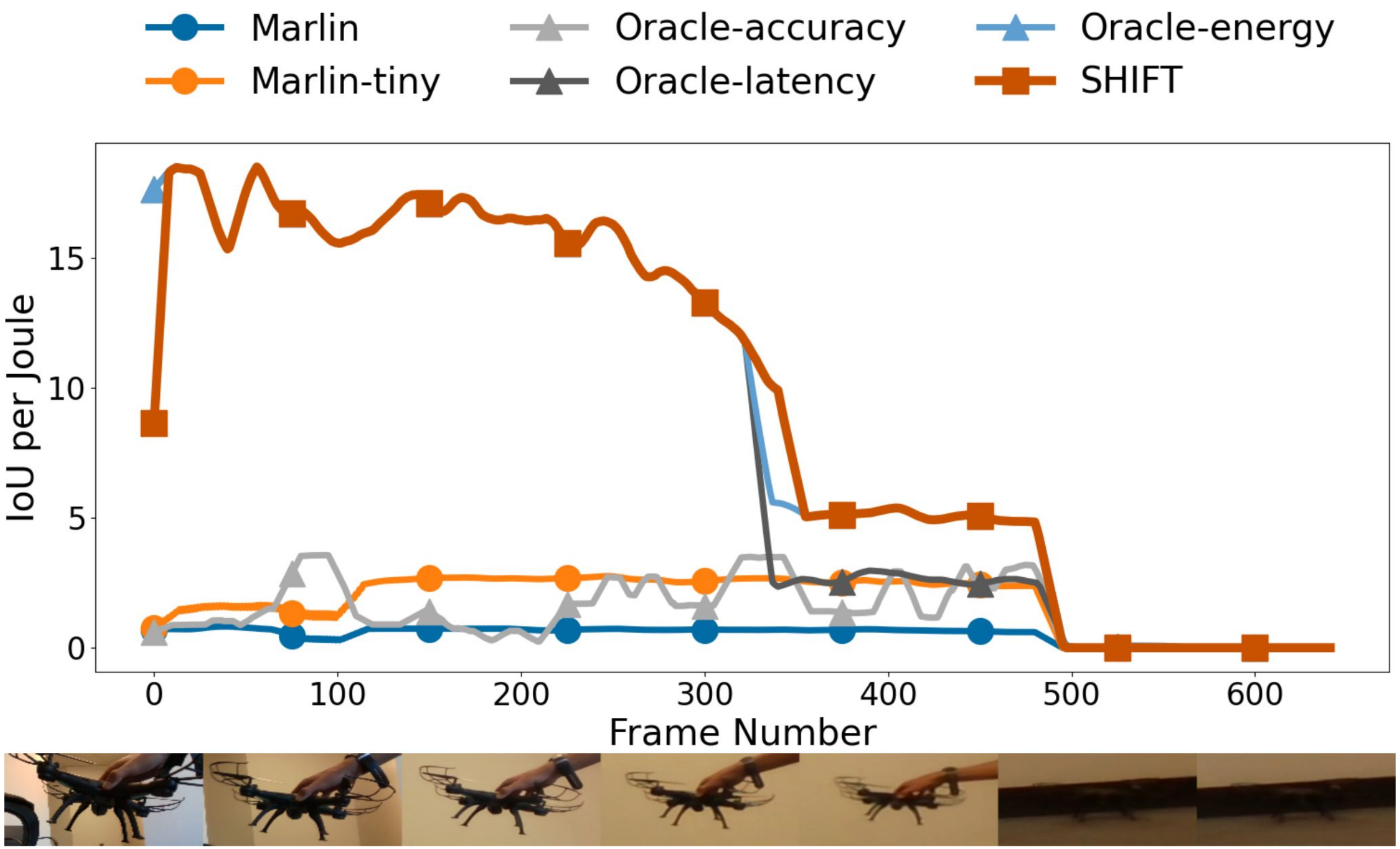
Scenario 2 - Slightly more complex



Scenario 3 - Complex



Scenario 4 - Simple Indoor

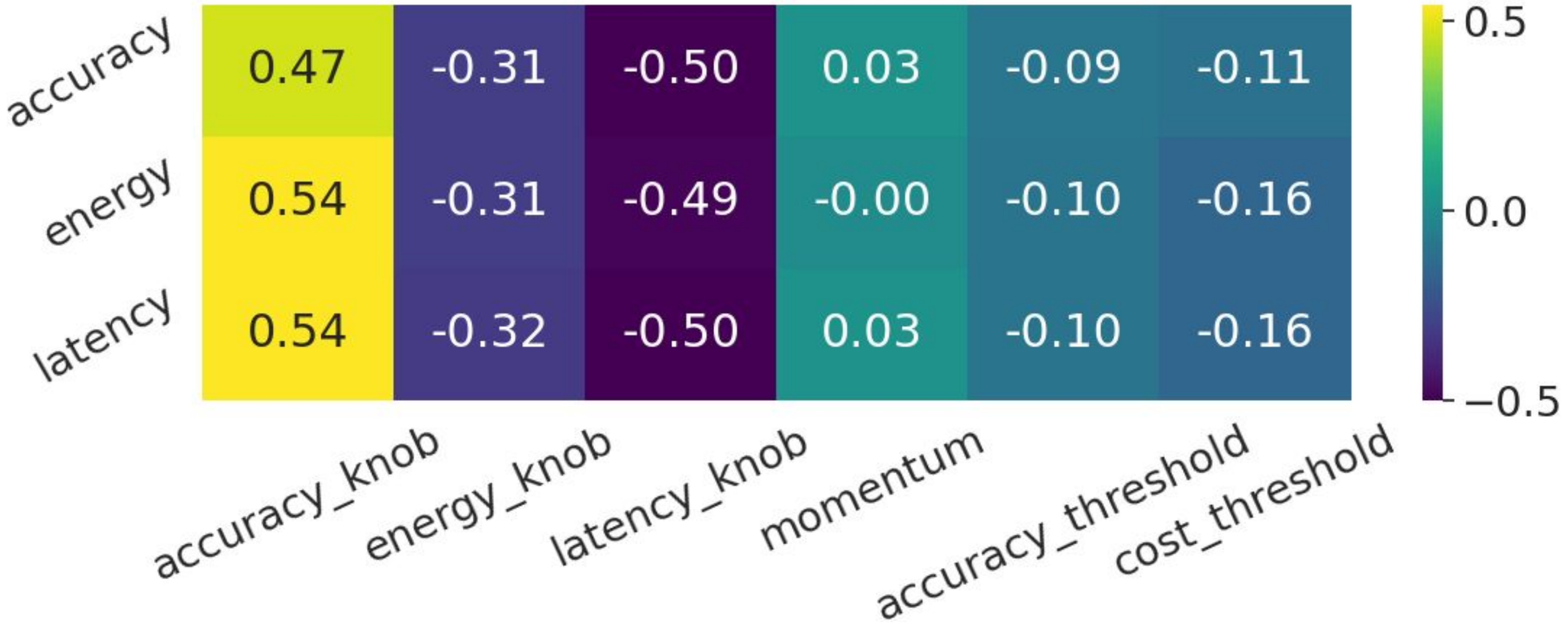


Overall Methodologies

Methodology	IoU	Time (s)	Energy (J)	Success Rate	Non-GPU	Model Swaps	Pairs Used
Marlin	0.614	0.132	1.201	74.0%	0%	0	1
Marlin Tiny	0.529	0.036	0.33	64.0%	0%	0	1
SHIFT	0.598	0.047	0.262	72.2%	68.7%	42	4.3
Oracle E	0.535	0.025	0.144	76.0%	31.5%	94	6.7
Oracle A	0.657	0.108	1.423	76.0%	44.9%	409	12.3
Oracle L	0.522	0.025	0.169	76.0%	11.3%	112	6.8

Maintained good results with fewer than half of the swaps and fewer allocated models than oracle methods.

Sensitivity Analysis



1. Knobs have correct relationship relative to each metric
2. Momentum (filtering results) does not have significant impact
3. Reducing cost threshold (using closer nodes only) has positive impacts

Conclusions

7.5x Energy usage improvement
2.8x Latency improvement
0.97x Accuracy performance
vs. YoloV7 on GPU

Thank you!

Any questions?